Resumen CLIPS

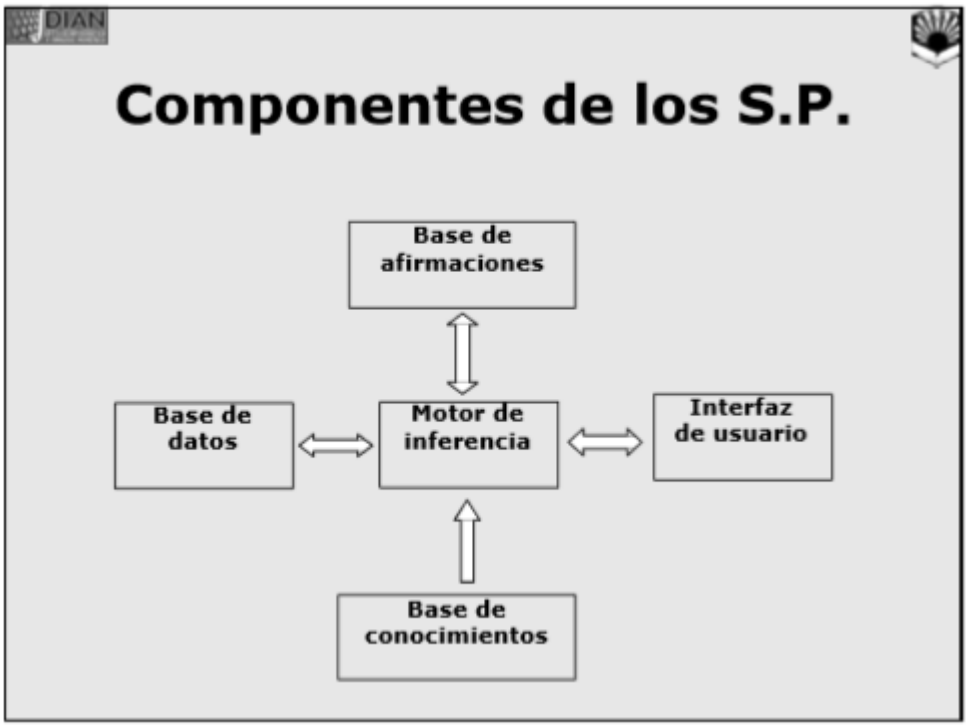
Tema 1. Introducción a los sistemas de producción

Sistemas de Producción

Las Reglas de Producción son reglas del tipo ***Si-Entonces***.

Las características de los Sistemas de producción:

- Se utilizan las reglas para examinar un conjunto de datos y solicitar nueva información hasta llegar a un diagnostico.
- También se denominan Sistemas basados en reglas



- Base de afirmaciones → Hechos.
- Base de conocimiento → Sentencias que aportan información al dominio.
- Base de datos → Datos (?)
- Interfaz de usuario → Ventanas y demás para que interactúe la peña
- Motor de inferencia → Extrae información a partir de la existente

Componentes

- **Base de conocimientos**
  Todos los hombres son animales
  Todos los animales respiran
- **Base de afirmaciones**
  Juan es un hombre
- **Inferencia**
  Juan respira

Reglas de producción

La estructura general de las reglas son: ***Antecedente*** → ***Consecuente***. Donde:

- **Antecedente**: Contiene las claúsulas que deben de cumplirse para que la regla pueda ejecutarse
- **Consecuente**: Indica las conclusiones que se deducen de las premisas o las acciones que el sistema debe realizar cuando ejecuta la regla.

Inferencia

- Una regla se ejecuta (dispara) cuando se cumple su antecedente
- Las reglas se ejecutan hacia adelante: **Si se satisface el antecedente se efectúan las acciones del consecuente**
- Tipos de encadenamiento de reglas:
    - Encadenamiento hacia adelante o **basado en datos**. → Este es el que utiliza CLIPS
    - Encadenamiento hacia detrás o basado en objetivos.

Control de Razonamiento Se encarga de seleccionar una regla cuando hay varias disponibles. Métodos de resolución de conflictos:

- Ordenación de reglas
- Añadir nuevas cláusulas relacionadas con las inferencias
- Control mediante *Agenda*
- Conjunto de Reglas

Tema 2. Visión general de CLIPS

Componentes de CLIPS

- Interprete
- Interfaz interactivo
- Facilidades para la depuración
- Elementos de la Shell
    - Lista de hechos
    - Base de conocimientos
    - Motor de inferencia
- Dirigido por datos: Las reglas pueden emparejar con objeto y hechos.

Tipos de datos primitivos

- Entero (*integer*)
- Reales (*float*)
- Simbolo (*symbol*)
- Cadena (*string*)
- Direccion externa (*external-address*)
- Direccion de hecho (*fact-address*)
- Nombre de instancia
- Direccion de instancia (*instance-address*)

Otros conceptos

- **Campo**: Valor que puede tomar una variable
- **Tipos de campos dependiendo del valor que puedan tomar**:
  - Monocampo: Tipos de datos primitvos
  - Multicampo: Varios valores uni-campo
- **Constante**
- **Variable**

Ejemplos

- `?<nombre>` → monocampo (ámbito local)
- `$?<nombre>` → multicampo (ámbito local)
- `?*<nombre>*` → ambos (ámbito global)

Funciones

- **Lenguaje externo**
  - Funciones definidas por el usuario
  - Funciones definidas por el sistema
- **Funciones de CLIPS**
  - Funciones
  - Funciones genericas
- **Tipos**
  - Órdenes: Ejecutan una acción
  - Funciones: Devuelven un valor
- **Notación prefija para llamada**
  - ((nombrefuncion argumentos) → `(+(* 3 4) 8 )`)

Constructores

- **defmodule** → Define un módulo
- **defrule** → Define una regla
- **deffacts** → Define un hecho o conjunto de estos
- **deftemplate** → Define una plantilla
- **defglobal** → Define una variable global
- **deffunction** → Define una función
- **defclass** → Define una clase
- **definstances** → Define una instancia
- [...]

Tema 3. Representación de Hechos en CLIPS

Representación de la información

Mediante:

- **Hechos**: Ordenados y no ordenados. Indice y dirección.
- **Objetos**: POO. Instancias de objetos.
- **Variables globales**: Constructor defglobal.

Hechos: Ordenes de uso

Órdenes de utilización de hechos:

- **assert** → Introduce datos en la base de hechos. *Se puede utilizar tambien deffacts*
- **facts** → Sirve para ver la base de hechos con formato `f-índice (hecho)`
- **undefacts** → Suprime los hechos insertados por una orden *deffacts*
- **ppdefacts** → Muestra la definición de un hecho definido por la orden *deffacts*
- **list-defacts** → Muestra los hechos definidos con *deffacts*
- **retract** → Elimina hechos de la base de hechos, se debe especificar el nombre o el indice del hecho se pueden eliminar varios hechos de golpe (retrat hecho1 hecho 2 ..) o eliminar todos con (retract *)
- **modify** → Modifica un hecho de la base de hechos
- **duplicate** → Duplica un hecho de la base de hechos
- **deftemplate** → Define una plantilla
- **ppdeftemplate** → Muestra una plantilla definida
- **undeftemplate** → Elimina una plantilla definida
- **deffacts** → Define un conjunto de hechos
- **reset** → Añade cada hecho especificado con *deffacts* en la lista de hechos o factlist. Tambien añade el hecho *initial-fact*. Tambien dice que borra hechos e inserta hecho especial (?).
- **clear** → Limpia la base de hechos. Reinicializa el indice de hechos a 0 y elimina la base de conocimiento

Hechos: comandos

```
(assert <hecho>+)
(facts [<inicio> [<final> [máximo]]])
(retract <índice>+ |*)
(modify <índice> <nueva-casilla>+)
(duplicate <índice> <nueva-casilla>+)
  <nueva-casilla>::= (<nombre> <valor>)
```

Hechos: Tipos y Ejemplos

- **Ordenados** → `(casa calle-nueva 32)`
- **No ordenados (Realizados mediante plantillas)**: `(coche (marca ford)(modelo fiesta))`
  - El orden de los campos no es importante
  - Se pueden modificar utilizando las órdenes (modify) y (duplicate)

Hechos: Dirección

```
(defrule comenzar
    ?h <- (iniciar-programa)
  =>
    retract(?h)
    (printout t "Iniciando..." crlf)
)
```

Hechos: Ejemplos de plantillas

```
  (deftemplate nombre-plantilla
      (slot nombre)
      (multislot apellidos)
      (slot DNI)
  )
```

Tema 4. Hechos definidos a partir de plantillas

Sintaxis del constructor deftemplate

```
(deftemplate persona
  (slot nombre)
  (multislot apellidos)
  (slot DNI))
```

Y su respectivo hecho con el cual empareja

```
(persona (nombre Juan)(apellidos Perez Perez) (DNI 43765873B))
```

Propiedades de los slots

- default-attribute: puede tener la necesidad o no de que el hecho tenga que tener ese campo a huevos
    - ?NONE: no es necesario un argumento para ese campon
    - ?DERIVE: es obligatorio el argumento

```
(deftemplate dato
  (slot x (default ?NONE))
  (slot y (default ?DERIVE))
)
```

- type: reestriccion de tipo para los slots, es decir, dicho slot debe llevar una variable de un determinado tipo (SYMBOL, FLOAT, INTEGER, NUMBER, STRING, ...)

```
(deftemplate dato
  (slot x (type STRING))
  (slot y (type NUMBER))
)
```

Dentro de esta reestriccion se puede incluir otra mas severa, acotando mas el rango

```
(deftemplate dato
  (slot x (allowed-symbols Pepe))
  (slot y (allowed-integer 6 5 9))
  (slot z (allowed-values "Ricardo" rico 99 1.e9))
)
```

Siendo posible acotar para los distintos tipos existentes en CLIPS. Siendo posible especificar rangos como se indica en el siguiente ejemplo

```
(deftemplate dato
  (slot x (allowed-symbols Pepe))
  (slot y (type integer)(range 8 90))
)
```

Tema 5. Reglas

El formato de una regla es *if-else*, es decir, si el antecedente es verdadero entonces se ejecuta el consecuente.

- Base de conocimiento: Conjunto de reglas que describen el problema a resolver
- Activación de reglas: Entidad patrón → hechos ordenados o plantillas, e instancias de clases
- Motor de inferencia: Comprueba el antecedente de las reglas y aplica el consecuente

Definición de reglas

Se realiza mediante el constructor defrule, y posee el siguiente formato:

```
(defrule regla1
  (frigorifico abierto)
  =>
  (comida mala)
)
```

- Consideraciones

- En caso de que haya una regla con el mismo nombre, la última machaca a la anterior
- No hay limite en el numero de elementos condicionales y acciones de una regla
- Puede no haber ningún elemento condicional en el antecedente y se usa automaticamente como elemento condicional
- Puede no haber ninguna acción en el consecuente, y la ejecución de la regla no tiene ninguna consecuencia

Ciclo de ejecución de reglas

- Las regla se ejecutan con el comando (run <máximo>)
- Si se ha alcanzado el numero máximo de disparos, se detiene la ejecución
- Se actualiza la agenda según la lista de hechos
- Se selecciona la instancia de regla a ejecutar de acuerdo a prioridades y estrategia de resolución de conflictos
- Se dispara la instancia seleccionada, se incrementa numero de disparos y se elimina de la agenda
- Volver al paso 2

Sintaxis del antecedente

El antecedente está compuesto por una serie de elementos condicionales. Son un conjunto de reestricciones que se usan para verificar si se cumple un campo o slot de una entidad patrón.

- EC patron → [Definido Abajo]
- EC test → Comprueba el valor devuelto por una función, si se satisface si la función devuelve un valor distinto de **FALSE**. No se satisface si la función devuelve un valor **FALSE**
    - Ejemplo:

```
(defrule diferencia
    (dato ?x)
    (valor ?y)
    (test (>= (abs(- ?x ?y)) 3))
    =>
)
```

- EC and → Se satisface si se satisface todos de los EC que lo componen. Se pueden mezclar ands y ors en el antecedente.
    - Ejemplo:

```
(defrule posibles-desayunos
  (tengo zumo-natural)
  (or (and (tengo pan)
  (tengo aceite))
  (and (tengo leche)
  (tengo cereales)))
=>
```

```
  (assert (desayuno sano))
)
```

- EC or → Se satisface si se satisface cualquiera de los EC que lo componen, si se satisface mas de uno, la regla se activará varias veces.
    - Ejemplo:

```
(defrule posibles-desayunos
  (tengo pan) (or (tengo mantequilla)
  (tengo aceite))
  =>
(assert (desayuno tostadas))
```

- EC not → Se satisface si no se satisface el EC que contiene. Donde solo puede negarse una vez
    - Ejemplo:

```
(defrule movil-sin-bateria
  (not (queda bateria))
  (not (hay enchufe))
=>
  (assert (estado panico))
)
```

```

```

- EC exists → Permiten comprobar si una serie de ECs se satisface por algún conjunto de hechos.
    - Ejemplo:

```
(defrule dia-salvado
  (objetivo salvar-el-dia)
  (exists (heroe (estado desocupado)))
=>
  (printout t "El día está salvado" crlf))
```

- EC forall → Permite comprobar si un conjunto de EC se satisface para toda ocurrencia de otro EC especificado. Se satisface, para toda ocurrencia del primer EC, se satisfacen los demás ECs
    - Ejemplo:

```

```

```
(defrule todos-limpios
  (forall (estudiante ?nombre)
    (lengua ?nombre)
    (matematicas ?nombre)
    (historia ?nombre))
  =>)
```

- EC logical → Asegura el mantenimiento de verdad para hechos creados mediante reglas que usan EC Logical.
    - Los hechos del antecedente proporcionan soporte lógico a los hechos creados en el consecuente.
    - Un hecho puede recibir soporte lógico de varios conjuntos distintos de hechos.
    - Un hecho permanece mientras permanezca alguno de los que lo soportan lógicamente.
    - Los ECs incluidos en el EC logical están unidos por un *and* implícito.
    - Puede combinarse con ECs *and*, *or* y *not*
    - Ejemplo:

```
(defrule puedo-pasar
  (logical (semaforo verde))
=>
  (assert (puedo pasar))
)
```

Sintaxis del antecedente

Siendo el siguiente formato el obligatorio para realizar la comparacion

- Para hechos ordenados → (...)
- Para hechos no ordenados → (... ())

Las distintas reestricciones son:

- Literales → Define el valor exacto que debe de poseer el campo, solo posee constantes.
- Comodines → Indican que cualquier valor en esa posición en válido para emparejar con la regla
    - Comodín monocampo → ?
    - Comodín multicampo → $?
- Variables → Almacena el valor de un campo para despues utilizarlo en otro elementos condicionales o consecuentes de una regla

    - Cuando la variabel aparece por 1ª vez, actua como un comodin, pero el valor queda **ligado al valor del campo**

- Conectivas → Permiten unir reestricciones y variables, utilizando los conectores lógicos and/or/not (&/|/~ → Siendo la prioridad de las conectivas ~/&/|)

    - Excepción: Si la primera restricción es una variable seguida de la conectiva &, la primera reestricción (la variable) se trata como una reestricción aparte
        - Ejemplo: ?x&rojo|azul equivale a ?x&(rojo|azul)

- Predicado → Se restringe un campo según el valor de verdad de una expresión lógica.

    - Esta se indica mediante dos puntos (:) seguidos de una llamada a función predicado.
    - La restricción se satisface si la función devuelve un valor dto. *FALSE*
    - Normalmente se usan junto a una restricción conectiva y a una variable
        - Ejemplo: ::= | ... | :

        ```
        (defrule predicado1
          (datos ?x&:(numberp ?x))
          =>
        )
        ```

    - Las funciones predicado que proporciona CLIPS son:
        - (evenp) → Comprueba que el argumento sea par
        - (floatp) → Comprueba que el argumento sea FLOAT
        - (integerp) → Comprueba que el argumento sea de tipo INTEGER
        - (numberp) → Comprueba que el argumento sea un numero
        - (oddp) → Comprueba que el argumento sea impar
        - (stringp) → Comprueba que el argumento sea de tipo STRING
        - (symbolp) → Comprueba que el argumento sea de tipo SYMBOL
    - Y las funciones de comparación:
        - (eq <expresión1><expresión2>+)
        - (neq <expresión1><expresión2>+)
        - (= <expresión1><expresión2>+)
        - (<> <expresión1><expresión2>+)
        - (> <expresión1><expresión2>+)
        - (< <expresión1><expresión2>+)
        - (>= <expresión1><expresión2>+)
        - (<= <expresión1><expresión2>+)

- Valores devueltos → Se usa el valor devuelto por una función para restringir un campo. Cuyo valor devuelto debe ser de uno de los tipos primitivos de datos y se sitúa en el patrón como si tratase de una restricción literal en las comparaciones.

    - Se implica mediante el caracter "="

    ```
    ```

```
(deftemplate datos
  (slot x)(slot y)
)

(defrule triple
  (datos (x ?x) (y =(* 3 ?x)))
  =>
)
```

- Direcciones de hechos: Para realizar modificaciones, duplicaciones o eliminaciones de hechos en el consecuente de una regla. Es necesario que en la regla se obtenga el indice del hecho sobre el que se desea actuar.
    - Ejemplo:

```
(defrule triple
  (datos (x ?x) (y =(* 3 ?x)))
  =>
)
```

Propiedades de una regla

- Los comandos disponibles para el manejo de reglas son:
    - (ppdefrule) → Visualiza una regla por pantalla
    - (list-defrules |*) → Muestra las reglas que hay
    - (rules) →
    - (undefrule |*) → Elimina la regla
- La declaración de propiedades se incluye tras el comentario y antes del antecedente
- Se indica mediante la palabra reservada *declare*
- Una regla puede tener una única sentencia *declare*
    - Prioridad
        - Se indica en la declaración de propiedades con la palabra reservada *salience*.
        - Puede tomar valores [-10000,10000], teniendo el valor 0 por defecto.
        - La prioridad puede evaluarse cuando se define la regla (por defecto), cuando se activa y en cada ciclo de ejecución.
        - Ejemplo:

```
(defrule triple
  (declare (salience 1000))
  (datos (x ?x) (y =(* 3 ?x)))
  =>
)
```

Tema 7. Variables globales

- El formato es → ?*nombre*, y se definen con *defglobal*
- Se pueden acceder desde dentro de los patrones
- El comando **bind** asigna un valor a una variable.
- El comando **reset** reinicializa su valor.
- Para eliminarlas se puede utilizar el comando **clear** o **undefglobal**
- Ejemplo:

```
(defglobal
  ?*x* = 3
  ?*y* = 6
)
```

- Diferencias con las variables locales:
  - No se puede usar de igual forma en el antecedente de una regla al asociarle un valor.
    - Ejemplo:

```
(defrule ejemplo ;incorrecto
  (fact ?*x*)
  => )

(defrule ejemplo ;correcto
  (fact ?y&:(= ?y ?*x*))
=>
  (bind ?*x* 3))
```

Tema 8. Modulos

- Con **defmodule** agrupa constructores (hechos,reglas,etc.) en módulos
- Los módulos se pueden borrar con la orden clear (y siempre debe de haber un módulo MAIN)
- Los módulos asociados con un nombre pueden especificarse explicitamente (*modulo::elemento*) o implicitamente (modulo actual o modulo activo).
- Con el comando **set-current-module** se activa un módulo diferente
- Los elementos deben de ser exportados o importados para poder ser usados por otros módulos

Ejecución de reglas

- Las formas de alterar el orden normal son:
  - Con la orden **focus** en el consecuente de una regla. Incluye un nuevo módulo en la pila de módulos activos y pasa el control a la agenda de este módulo.
  - Con la orden **return** en el consecuente de una regla. Esto provocaría que se pasaría como módulo activo el siguiente módulo activo el siguiente módulo de la pila de módulos activos y

pasa el control a su agenda.

INLUIR ALGUN EJEMPLILLO

Tema 9. Funciones

Con el constructor deffunction permite crear nuevas funciones dentro de CLIPS, compuesta por:

- Nombre
- Comentario (opcional)
- Cero o mas parametro regulares
- Un parametro comodin opcional que gestiona un número variable de argumentos
- Una secuencia de acciones, o expresiones, que se ejecutaran de forma secuencial cuando se llame a la función definida mediante este constructor.
- El formato es el siguiente:

```
(deffunction <nombre> [<comentario>]
  (<parametro-regular>*
  [<parametro-comodin>])
<accion>*

===================================

<paramatro-regular> ::= <variable-de-campo-simple>

===================================

<paramatro-comodin> ::= <variable-multicampo>
```

- Conceptos:
    - Una función debe de tener un nombre único
    - Una función debe de ser declarada antes de ser llamada
    - Tambien pueden llamarse a si misma y ser recursiva

Tema 11. Acciones y funciones

- Funciones de predicado:
    - (integerp <expresión>)
    - (floatp <expresión>)
    - (numberp <expresión>)
    - (symbolp <expresión>)
    - (stringp <expresión>)
    - (lexemep <expresión>)

    - (evenp <expresión>)
    - (oddp <expresión>)
    - (multifieldp <expresión>)

- Otras funciones de predicado:

    - (eq <expresión><expresión1>+)
    - (neq <expresión><expresión1>+)
    - (= <expresión><expresión1>+)
    - (<> <expresión><expresión1>+)
    - (< <expresión><expresión1>+)
    - (<= <expresión><expresión1>+)
    - (> <expresión><expresión1>+)
    - (>= <expresión><expresión1>+)

- Y venga funciones de predicado, que son gratis:

    - (and +)
    - (or +)
    - (not)

- Funciones multicampo:

    - (create$ <expresión>*)
    - (length$)
    - (nth$)
    - (member$ <expresión>)
    - (subsetp)
    - (subseq$)
    - (first$)
    - (rest$)
    - (explode$)
    - (implode$
    - (insert$ <expresión>+)
    - (replace$ <expresión>+)
    - (delete$

- Funciones de cadena:

    - (str-cat <expresión>*)
    - (sym-cat <expresión>*)
    - (str-length <expr-cadena-o-símbolo>)
    - (sub-string)
    - (str-compare <expr-cadena-o-símbolo> <expr-cadena-o-símbolo>)

- Funciones matemáticas

    - (+ +)
    - (- +)
    - (* +)

    - (/ +)
    - (div +)
    - (mod)
    - (sqrt)
    - (**)
    - (round)
    - (abs)
    - (max +)
    - (min +

- Funciones procedurales
    - (bind <expresión>*)
    - *if-esle*

```
(if <expresión>
  then <acción>*
  [else <acción>\*])
```

    - *switch-case* (switch [])

      ::= (case <expr-comparación> then <acción>)

      ::= (default <acción>)

    - Bucles
        - (while <expresión> [do] <acción>*)
        - Bucle raro:

```
(loop-for-count <rango> [do] <acción>\*)
<rango> ::=
  (<variable> <índice-final>) |
  (<variable> <índice-inicio> <índice-final>)

<índice-final> | <índice-inicio> ::= <expr-entera>
<índice-final> ::= <expr-entera>
(return [<expresion>])
(break)
```

Extra preguntas cuestionario SIN

.

Pregunta 1

Sin responder aún

Puntúa como 1,00

Marcar pregunta

En un ciclo de ejecución del motor de inferencia todas las reglas que estén activadas en la agenda se disparan y sus acciones son ejecutadas.

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 2

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

Para que una regla se active y pase a la agenda se deben de satisfacer todos los elementos condicionales del antecedente de la regla.

Seleccione una:

- Verdadero
- Falso

. . .

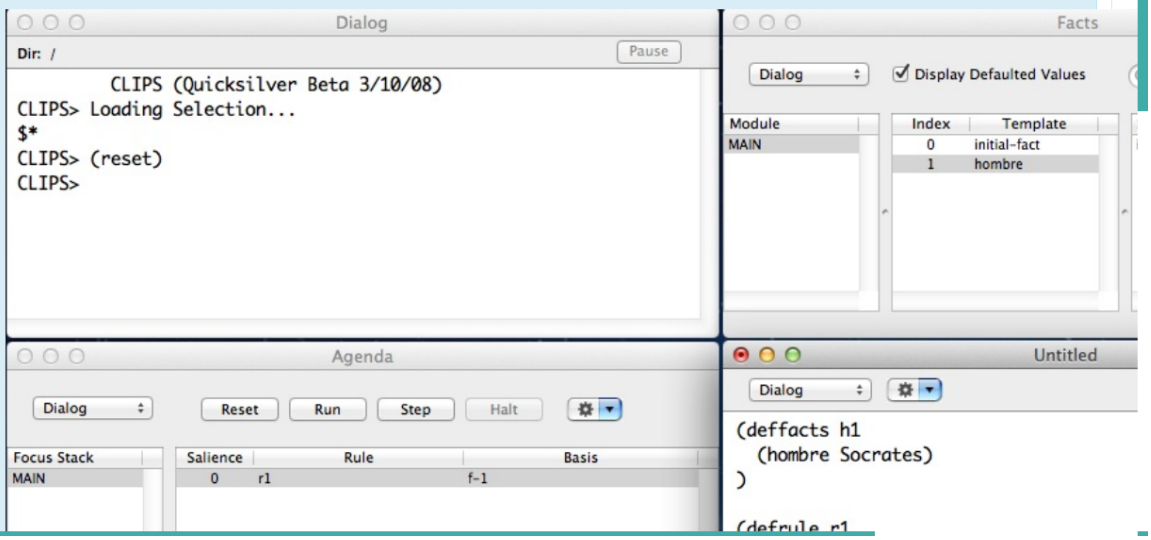
Pregunta 3

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

Ls siguiente figura muestra un entrono de clips. El programa escrito permite al sistema inferir que Socrates es mortal.



. . .

Pregunta 4

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

Se dice que clips es un programa dirigido por datos. La justificación es:

Seleccione una:

- a. Los datos estimulan a las reglas. Si los datos o hechos emparejan con los elementos condicionales del antecedente de la regla, la regla se activa y se ubica en la agenda
- b. Sin datos no es posible la ejecución de un programa
- c. La afirmación es falsa

. . .

Pregunta 5

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

En el siguiente programa de clips:

```
(deffacts h1
  (hombre Socrates)
)
(defrule r1
(hombre ?x)
=>
(assert (mortal ?x))
)
```

El mecanismo *de inferencia se denomina comparación de patrones.* El ingeniero debe de programar este mecanismo para cada programa de clips que desee ejecutar.

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 6

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

En el siguiente programa de clips:

```
(deffacts h1
  (hombre Socrates)
)
(defrule r1
(mortal ?x)
=>
(printout t ?x " es mortal" crlf )
)

(defrule r2
(hombre ?x)
=>
(assert (mortal ?x))
)
```

El motor de inferencia ejecutaría primero la regla r1 y después la regla r2

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 7

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

En la siguiente regla

```
(defrule r1
(hombre Socrates)
=>
(assert (mortal ?x))
)
```

(hombre Socrates) representa un hecho ordenado

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 8

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

En las siguientes frases:

1) Todos los hombre son mortales

2) Sócrates es un hombre

La primera corresponde a conocimiento y en un formalismo de representación de sistema basado en reglas (SBR) se representará a través de una regla

La segunda frase corresponde a una afirmación o a un hecho y en un SBR podemos representarlo a través de un hecho.

A través de la programación orientada a objetos estas dos frases podrían trasladarse a un programa que permitiera inferir automáticamente que Sócrates es mortal. El mecanismo de inferencia que se utilizaría sería el de la ***herencia***

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 9

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

La orden ***(reset)*** carga los hechos de los constructores ***deffacts*** en la base de hechos y prepara al sistema para la ejecución

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 10

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

La orden ***(clear)*** borra completamente la memoria de trabajo de clips tanto de la base de hechos como la base de conocimiento.

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 11

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

El código del siguiente programa es correcto:

```
(deffacts h1 ;Constructor de hechos
  (n 0) ; Hecho ordenado
  )

(defrule r1
?f<-(n  ?x) ; Elemento condicional patrón (ECP)
          ; A la variable ?x se le ligará valores de los hechos que emparejen
          ; A la variable ?f se le liga la dirección de hecho con el que
          ; empareje el ECP

(test (< ?x 10)); Elemento condicional test
=>
(printout t "n= " ?y crlf) ;Acción de imprimir
(assert (n (+ ?x 1))); Afirmación de un hecho nuevo (n resultado-de-la-suma)
(retract ?f) ;Elimina el hecho cuya dirección está en la variable ?f
)
```

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 12

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

En las siguientes frases:

1) Todos los hombre son mortales

2) Sócrates es un hombre

La primera corresponde a conocimiento y en un formalismo de representación de sistema basado en reglas (SBR) se representará a través de una regla

La segunda frase corresponde a una afirmación o a un hecho y en un SBR podemos representarlo a través de un hecho.

Un programa en clips como el siguiente permitiría realizar inferencias:

```
(deffacts h1
  (hombre Socrates)
)
(defrule r1
(hombre ?x)
=>
(assert (mortal ?x))
)
```

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 6

Sin responder aún

Puntúa como 1,00

Marcar pregunta

En el siguiente programa de clips:

```
(deffacts h1
  (hombre Socrates)
)
(defrule r1
(mortal ?x)
=>
(printout t ?x " es mortal" crlf )
)

(defrule r2
(hombre ?x)
=>
(assert (mortal ?x))
)
```

El motor de inferencia ejecutaría primero la regla r1 y después la regla r2

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 15

Sin responder aún

Puntúa como 1,00

Marcar pregunta

En la siguiente regla:

```
(defrule encontrar-datos
     (datos 1 azul rojo)
=>
 )
```

el elemento condicional patrón se comparará con hechos ordennados y su primer campo debe ser un símbolo.

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 1

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

En el antecedente de una regla

Seleccione una o más de una:

- [x] a. Hay 8 tipos de EC que son:
  - -EC patrón
  - -EC test
  - -EC and
  - -EC or
  - -EC not
  - -EC exists
  - -EC forall
  - -EC logical
- [ ] b. Pueden incluirse acciones que afirmen hechos
- [ ] c. Hay diferentes tipos de elementos condicionales patrón

. . .

Pregunta 2

Sin responder aún

Puntúa como 1,00

Marcar pregunta

En los elementos condicionales patrón cuando se usan restricciones con variables:

Seleccione una o más de una:

- [x] a. Las variables son locales a la regla
- [ ] b. La segunda vez que aparezca la variable en la regla vuelve a ligarse con un valor del siguiente hecho en la base de hechos que empareje.
- [ ] c. La liguadura de un valor a una variable se mantiene únicamente en el ámbito de la regla

. . .

Pregunta 3

Sin responder aún

Puntúa como 1,00

Marcar pregunta

En el siguiente programa

```
 (deftemplate datos-B
  (slot valor)
)

(deffacts h1
(datos-A verde)
(datos-A azul)
(datos-B (valor rojo))
(datos-B (valor azul))
)

(defrule r1 (datos-A ~azul) => )
(defrule r2 (datos-B (valor ~rojo&~verde)) => )
(defrule r3 (datos-B (valor verde|rojo)) => )
```

Seleccione una o más de una:

- [ ] a. La regla r1 se activará dos veces
- [x] b. La regla r2 no se activaría
- [ ] c. La regla r3 se activaría una vez

. . .

Pregunta 4

Sin responder aún

Puntúa como 1,00

Marcar pregunta

En el siguiente programa

```
(deftemplate datos
  (slot x)
  (slot y)
)
(deffacts hechos
 (datos (x 2) (y 4))
 (datos (x 3) (y 9))
)

(defrule r1
  (datos (x ?x) (y =(* 2 ?x)))
  =>
)
```

Seleccione una o más de una:

- [ ] a. La regla ***r1*** se activaría dos veces
- [ ] b. En la regla r1 se muestra un elemento condicioanal patrón con una restricción en la que se invoca a una función que devuelve un valor.
- [x] c. La regla ***r1*** se acrivaría por el hecho ***(datos (x 2) (y 4))***
- [ ] d. La regla r1 no se activaría

. . .

Pregunta 5

Sin responder aún

Puntúa como 1,00

Marcar pregunta

Con los siguientes hechos cuantas veces se activaría la siguiente regla? Responda uno de los posibles valores (1, 2, 3, 4)

```
f-1 (datos 1.0 azul "rojo")
f-2 (datos 1 azul)
f-3 (datos 1 azul rojo)
f-4 (datos 1 azul ROJO)
f-5 (datos 1 azul rojo 6.9)

(defrule encontrar-datos
   (datos ? azul rojo $?) => )
```

Respuesta: 3

. . .

Pregunta 6

Sin responder aún

Puntúa como 1,00

Marcar pregunta

En el siguiente programa

```
(deffacts hechos
  (comidas carne huevos pescado)
)

(defrule r1
  (comidas $?antes xxx $?despues)
 =>
  (printout t ?x crlf)
)
```

¿Qué habría que escribir en lugar ***xxx*** para que se mostrara por pantalla todas las comidas?

Respuesta: huevos

. . .

Pregunta 7

Sin responder aún

Puntúa como 1,00

Marcar pregunta

En el siguiente código

```
(deffacts hechos
  (comidas carne huevos pescado)
)

(defrule r1
  (comidas xxxa ?x xxxq)
 =>
  (printout t ?x crlf)
)
```

*qué habría que escribir en lugar de **xxx** para que se mostrara por pantalla todas las comidas al ejecutsrse el programa*

Respuesta: carne huevos

Página anterior Siguiente página

. . .

Pregunta 8

Sin responder aún

Puntúa como 1,00

Marcar pregunta

Si tenemos una plantilla

```
(deftemplate persona
    (multislot nombre)
     (slot edad))
```

El patrón *(persona (nombre Juan)) emparejaría con el hecho (persona (nombre Juan))*

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 9

Sin responder aún

Puntúa como 1,00

Marcar pregunta

En la siguiente regla:

```
(defrule encontrar-datos
     (datos 1 azul rojo)
 =>
 )
```

el elemento condicional patrón contiene sólo restricciones literales.

Seleccione una:

- Verdadero
- Falso

Página anterior Siguiente página

. . .

Pregunta 10

Sin responder aún

Puntúa como 1,00

Marcar pregunta

La siguiente regla

(defrule encontrar-datos
(datos 1 azul rojo) =>)

con los siguientes hechos no se activaría

f-0 (initial-fact)
f-1 (datos 1.0 azul "rojo")
f-2 (datos 1 azul)
f-3 (datos 1 azul rojo)
f-4 (datos 1 azul ROJO)
f-5 (datos 1 rojo azul)
f-6 (datos 1 azul rojo 6.9)

Seleccione una:

- Verdadero
- Falso

Página anterior | Siguiente página

. . .

Pregunta 11

Sin responder aún

Puntúa como 1,00

Marcar pregunta

El siguiente ECP *(dato $? VERDE $?)* no emparejaría con todos los siguientes hechos:

(dato VERDE)
(dato VERDE rojo azul)
(dato rojo VERDE azul)
(dato rojo azul VERDE)
(dato VERDE azul VERDE)

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 12

Sin responder aún

Puntúa como 1,00

Marcar pregunta

La siguiente regla dará error en tiempo de ejecución

(defrule prueba
=>
(printout t ?x crlf))

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 13

Sin responder aún

Puntúa como 1,00

Marcar pregunta

Los elementos condicionales patrón utilizan los conectores lógicos & *(and)*, | *(or)*, ~ *(not)*
y el orden de precedencia entre ellos es ~, &, |

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 14

Sin responder aún

Puntúa como 1,00

Marcar pregunta

La siguiente regla muestra un elemento condicional patrón con una restricción predicado

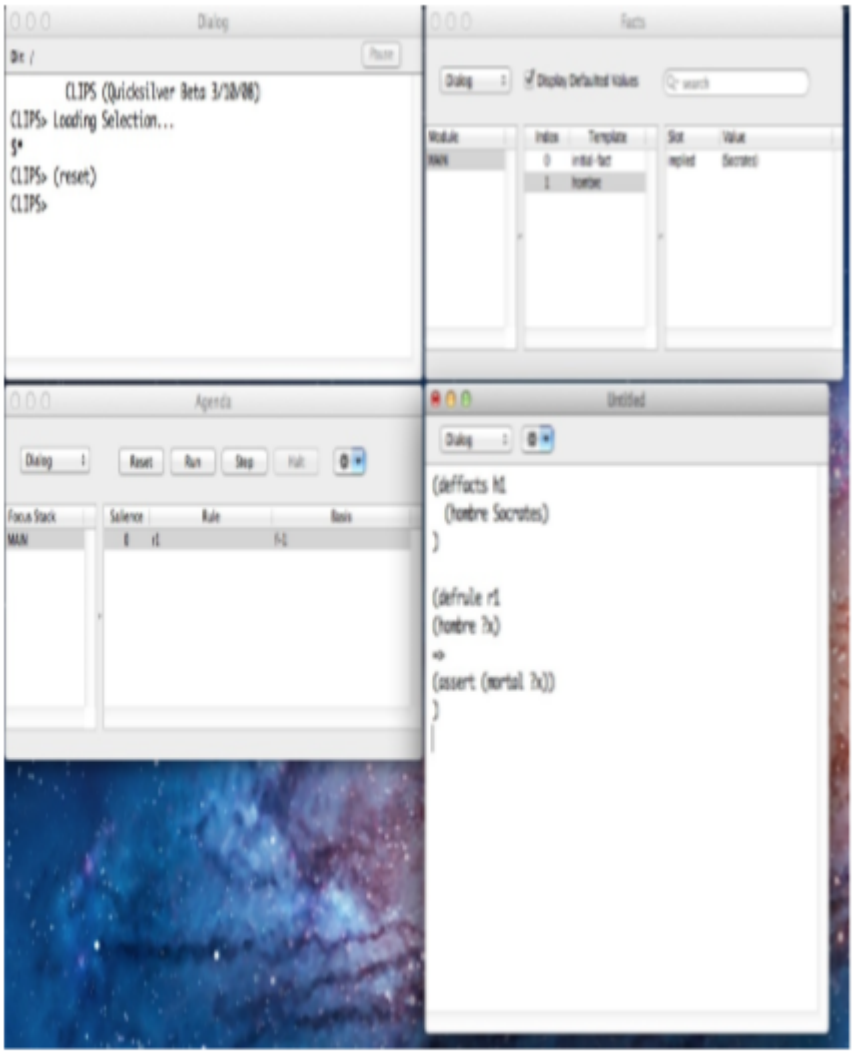
```
(defrule r1
 (datos ?x&:(numberp ?x))
=>
)
```

Seleccione una:

- Verdadero
- Falso

Algunas soluciones

Ls siguiente figura muestra un entrono de clips. El programa escrito permite al sistema inferir que Socrates es mortal.



Seleccione una:

- Verdadero
- Falso

En la figura se muestra la estructura general de un programa básico en CLIPS. La regla r1 está activa porque el patrón (hombre ?x) empareja o encaja con el hecho (hombre Socrates)

La respuesta correcta es 'Verdadero'

. . .

Pregunta 2

Correcta

Puntúa 1,00 sobre 1,00

La orden ***(clear)*** borra completamente la memoria de trabajo de clips tanto de la base de hechos como la base de conocimiento.

Seleccione una:

- Verdadero
- Falso

. . .

Pregunta 3

Correcta

Puntúa 1,00 sobre 1,00

En un ciclo de ejecución del motor de inferencia todas las reglas que estén activadas en la agenda se disparan y sus acciones son ejecutadas.

Seleccione una:

- Verdadero
- Falso

En cada ciclo de ejecución sólo se dispara una regla, la que esté primero en la agenda. En el siguiente ciclo las reglas son de nuevo evaluadas y posicionadas en la agenda.

La respuesta correcta es 'Falso'

. . .

Pregunta 4

Correcta

Puntúa 1,00 sobre 1,00

En las siguientes frases:

1) Todos los hombre son mortales

2) Sócrates es un hombre

La primera corresponde a conocimiento y en un formalismo de representación de sistema basado en reglas (SBR) se representará a través de una regla

La segunda frase corresponde a una afirmación o a un hecho y en un SBR podemos representarlo a través de un hecho.

Un programa en clips como el siguiente permitiría realizar inferencias:

```
(deffacts h1
  (hombre Socrates)
)

(defrule r1
(hombre ?x)
=>
(assert (mortal ?x))
)
```

Seleccione una:

- Verdadero
- Falso

La respuesta correcta es 'Verdadero'

. . .

Pregunta 5

Incorrecta

Puntúa 0,00 sobre 1,00

En la siguiente regla

(defrule r1
(hombre Socrates)
=>
(assert (mortal ?x))
)

(hombre Socrates) representa un hecho ordenado

Seleccione una:

- Verdadero
- Falso

(hombre Socrates) representa un elemento condicional ya que forma parte del antecedente de la regla

La respuesta correcta es 'Falso'

. . .

Pregunta 6

Correcta

Puntúa 1,00 sobre 1,00

En el siguiente programa de clips:

```
(deffacts h1
  (hombre Socrates)
)
(defrule r1
(mortal ?x)
=>
(printout t ?x " es mortal" crlf )
)

(defrule r2
(hombre ?x)
=>
(assert (mortal ?x))
)
```

El motor de inferencia ejecutaría primero la regla r1 y después la regla r2

Seleccione una:

- Verdadero
- Falso

El orden que que estén escritas las reglas no influye en el orden de ejecución. Primero se activaría la regla r2 porque es la única que tiene hechos que encajen con sus elementos condicionales patrón.

La respuesta correcta es 'Falso'

. . .

Pregunta 7

Incorrecta

Puntúa 0,00 sobre 1,00

En las siguientes frases:

1) Todos los hombre son mortales

2) Sócrates es un hombre

La primera corresponde a conocimiento y en un formalismo de representación de sistema basado en reglas (SBR) se representará a través de una regla

La segunda frase corresponde a una afirmación o a un hecho y en un SBR podemos representarlo a través de un hecho.

A través de la programación orientada a objetos estas dos frases podrían trasladarse a un programa que permitiera inferir automáticamente que Sócrates es mortal. El mecanismo de inferencia que se utilizaría sería el de la ***herencia***

Seleccione una:

- Verdadero
- Falso

Esto demuestra que la información y el conocimiento pueden representarse usando diferentes formalismos de representación simbólica.

La respuesta correcta es 'Verdadero'

. . .

Pregunta 8

Incorrecta

Puntúa 0,00 sobre 1,00

El código del siguiente programa es correcto:

```
(deffacts h1 ;Constructor de hechos
  (n 0) ; Hecho ordenado
  )

(defrule r1
?f<-(n  ?x) ; Elemento condicional patrón (ECP)
          ; A la variable ?x se le ligará valores de los hechos que emparejen
          ; A la variable ?f se le liga la dirección de hecho con el que
          ; empareje el ECP

(test (< ?x 10)); Elemento condicional test
=>
(printout t "n= " ?y crlf) ;Acción de imprimir
(assert (n (+ ?x 1))); Afirmación de un hecho nuevo (n resultado-de-la-suma)
(retract ?f) ;Elimina el hecho cuya dirección está en la variable ?f
)
```

Seleccione una:

- Verdadero
- Falso

La variable ***?y*** no puede usarse ya que no se le ha asociado ningún valor

La variable ***?y*** no puede usarse ya que no se le ha asociado ningún valor

La respuesta correcta es 'Falso'

. . .

Pregunta 9

Correcta

Puntúa 1,00 sobre 1,00

La orden ***(reset)*** carga los hechos de los constructores ***deffacts*** en la base de hechos y prepara al sistema para la ejecución

Seleccione una:

- Verdadero
- Falso

La respuesta correcta es 'Verdadero'

. . .

Pregunta 10

Incorrecta

Puntúa 0,00 sobre 1,00

En el siguiente programa de clips:

```
(deffacts h1
  (hombre Socrates)
)
(defrule r1
(hombre ?x)
=>
(assert (mortal ?x))
)
```

El mecanismo *de inferencia se denomina comparación de patrones.* El ingeniero debe de programar este mecanismo para cada programa de clips que desee ejecutar.

Seleccione una:

- Verdadero
- Falso

Al mecanismo de inferencia se le denomina ***comparación de patrones***. Este mecanimso y viene programado en Clips por defecto y se usa automáticamente.

La respuesta correcta es 'Falso'

. . .

Pregunta 11

Correcta

Puntúa 1,00 sobre 1,00

Para que una regla se active y pase a la agenda se deben de satisfacer todos los elementos condicionales del antecedente de la regla.

Seleccione una:

- Verdadero
- Falso

La respuesta correcta es 'Verdadero'

. . .

Pregunta 12

Correcta

Puntúa 1,00 sobre 1,00

Se dice que clips es un programa dirigido por datos. La justificación es:

Seleccione una:

- a. Los datos estimulan a las reglas. Si los datos o hechos emparejan con los elementos condicionales del antecedente de la regla, la regla se activa y se ubica en la agenda
- b. Sin datos no es posible la ejecución de un programa
- c. La afirmación es falsa

Respuesta correcta

La respuesta correcta es: Los datos estimulan a las reglas. Si los datos o hechos emparejan con los elementos condicionales del antecedente de la regla, la regla se activa y se ubica en la agenda

. . .

Pregunta 1

Parcialmente correcta

Puntúa 0,50 sobre 1,00

En el antecedente de una regla

Seleccione una o más de una:

- a. Hay 8 tipos de EC que son:
  –EC patrón
  –EC test
  –EC and
  –EC or
  –EC not
  –EC exists
  –EC forall
  –EC logical
- b. Pueden incluirse acciones que afirmen hechos
- c. Hay diferentes tipos de elementos condicionales patrón

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.
Las respuestas correctas son: Hay 8 tipos de EC que son:
–EC patrón
–EC test
–EC and
–EC or
–EC not
–EC exists
–EC forall
–EC logical, Hay diferentes tipos de elementos condicionales patrón

. . .

Pregunta 2

Parcialmente correcta

Puntúa 0,50 sobre 1,00

En los elementos condicionales patrón cuando se usan restricciones con variables:

Seleccione una o más de una:

- [x] a. Las variables son locales a la regla
- [ ] b. La segunda vez que aparezca la variable en la regla vuelve a ligarse con un valor del siguiente hecho en la base de hechos que empareje.
- [ ] c. La liguadura de un valor a una variable se mantiene únicamente en el ámbito de la regla

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.
Las respuestas correctas son: La liguadura de un valor a una variable se mantiene únicamente en el ámbito de la regla, Las variables son locales a la regla

. . .

Pregunta 3

Incorrecta

Puntúa 0,00 sobre 1,00

En el siguiente programa

```
(deftemplate datos-B
  (slot valor)
)

(deffacts h1
(datos-A verde)
(datos-A azul)
(datos-B (valor rojo))
(datos-B (valor azul))
)

(defrule r1 (datos-A ~azul) => )
(defrule r2 (datos-B (valor ~rojo&~verde)) => )
(defrule r3 (datos-B (valor verde|rojo)) => )
```

Seleccione una o más de una:

- [ ] a. La regla r1 se activará dos veces
- [x] b. La regla r2 no se activaría
- [ ] c. La regla r3 se activaría una vez

Respuesta incorrecta.

La respuesta correcta es: La regla r3 se activaría una vez

. . .

Pregunta 4

Parcialmente correcta

Puntúa 0,50 sobre 1,00

En el siguiente programa

```
(deftemplate datos
  (slot x)
  (slot y)
)
(deffacts hechos
 (datos (x 2) (y 4))
 (datos (x 3) (y 9))
)

(defrule r1
  (datos (x ?x) (y =(* 2 ?x)))
  =>
)
```

Seleccione una o más de una:

- [ ] a. La regla ***r1*** se activaría dos veces
- [ ] b. En la regla r1 se muestra un elemento condicioanal patrón con una restricción en la que se invoca a una función que devuelve un valor.
- [x] c. La regla ***r1*** se acrivaría por el hecho ***(datos (x 2) (y 4))***
- [ ] d. La regla r1 no se activaría

Respuesta parcialmente correcta.

Ha seleccionado correctamente 1.
Las respuestas correctas son: La regla ***r1*** se acrivaría por el hecho ***(datos (x 2) (y 4))***, En la regla r1 se muestra un elemento condicioanal patrón con una restricción en la que se invoca a una función que devuelve un valor.

. . .

Pregunta 5

Incorrecta

Puntúa 0,00 sobre 1,00

Con los siguientes hechos cuantas veces se activaría la siguiente regla? Responda uno de los posibles valores (1, 2, 3, 4)

```
f-1 (datos 1.0 azul "rojo")
f-2 (datos 1 azul)
f-3 (datos 1 azul rojo)
f-4 (datos 1 azul ROJO)
f-5 (datos 1 azul rojo 6.9)

(defrule encontrar-datos
   (datos ? azul rojo $?) => )
```

Respuesta: 3

La respuesta correcta es: 2

. . .

Pregunta 6

Incorrecta

Puntúa 0,00 sobre 1,00

En el siguiente programa

```
(deffacts hechos
  (comidas carne huevos pescado)
)

(defrule r1
  (comidas $?antes xxx $?despues)
 =>
  (printout t ?x crlf)
)
```

¿Qué habría que escribir en lugar ***xxx*** para que se mostrara por pantalla todas las comidas?

Respuesta: huevos

La respuesta correcta es: ?x

. . .

Pregunta 7

Incorrecta

Puntúa 0,00 sobre 1,00

En el siguiente código

```
(deffacts hechos
  (comidas carne huevos pescado)
)

(defrule r1
  (comidas xxxa ?x xxxq)
 =>
  (printout t ?x crlf)
)
```

*qué habría que escribir en lugar de **xxx** para que se mostrara por pantalla todas las comidas al ejecutsrse el programa*

Respuesta: carne huevos

La respuesta correcta es: $?

. . .

Pregunta 8

Correcta

Puntúa 1,00 sobre 1,00

Si tenemos una plantilla

```
(deftemplate persona
    (multislot nombre)
     (slot edad))
```

El patrón *(persona (nombre Juan)) emparejaría con el hecho (persona (nombre Juan))*

Seleccione una:

- Verdadero
- Falso

La respuesta correcta es 'Verdadero'

. . .

Pregunta 9

Correcta

Puntúa 1,00 sobre 1,00

En la siguiente regla:

(defrule encontrar-datos
(datos 1 azul rojo)
=>
)

el elemento condicional patrón contiene sólo restricciones literales.

Seleccione una:

- Verdadero
- Falso

La respuesta correcta es 'Verdadero'

. . .

Pregunta 10

Correcta

Puntúa 1,00 sobre 1,00

La siguiente regla

(defrule encontrar-datos
(datos 1 azul rojo) =>)

con los siguientes hechos no se activaría

f-0 (initial-fact)
f-1 (datos 1.0 azul "rojo")
f-2 (datos 1 azul)
f-3 (datos 1 azul rojo)
f-4 (datos 1 azul ROJO)
f-5 (datos 1 rojo azul)
f-6 (datos 1 azul rojo 6.9)

Seleccione una:

- Verdadero
- Falso

La respuesta correcta es 'Falso'

. . .

Pregunta 11

Correcta

Puntúa 1,00 sobre 1,00

El siguiente ECP *(dato $? VERDE $?)* no emparejaría con todos los siguientes hechos:

(dato VERDE)
(dato VERDE rojo azul)
(dato rojo VERDE azul)
(dato rojo azul VERDE)
(dato VERDE azul VERDE)

Seleccione una:

- Verdadero
- Falso

La respuesta correcta es 'Falso'

. . .

Pregunta 12

Incorrecta

Puntúa 0,00 sobre 1,00

La siguiente regla dará error en tiempo de ejecución

(defrule prueba
=>
(printout t ?x crlf))

Seleccione una:

- Verdadero
- Falso

La respuesta correcta es 'Verdadero'

. . .

Pregunta 13

Correcta

Puntúa 1,00 sobre 1,00

Los elementos condicionales patrón utilizan los conectores lógicos & *(and)*, | *(or)*, ~ *(not)*
y el orden de precedencia entre ellos es ~, &, |

Seleccione una:

- Verdadero
- Falso

La respuesta correcta es 'Verdadero'

. . .

Pregunta 14

Correcta

Puntúa 1,00 sobre 1,00

La siguiente regla muestra un elemento condicional patrón con
una restricción predicado

(defrule r1
(datos ?x&:(numberp ?x))
=>
)

Seleccione una:

- Verdadero
- Falso

La respuesta correcta es 'Verdadero'

. . .

Pregunta 1

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

Siempre y cuando la variable global `?*x*` esté definida, la siguiente expresión sería válida:

```
(bind ?*x* 23)
```

Seleccione una:

- a. No contestar
- b. Falso
- c. Verdadero

Siguiente página

. . .

Pregunta 2

Incorrecta

Puntúa -0,50 sobre 1,00

Marcar pregunta

El siguiente constructor contiene, al menos, un error y no sería válido:

```
(defglobal MAIN
  ?*y1* = -9
  ?*y2* = ?*y1*
)
```

Seleccione una:

- a. Falso
- b. Verdadero ✗
- c. No contestar

Respuesta incorrecta.

La sintaxis es correcta: crea dos variables globales en el módulo MAIN (que siempre existe) y crea dos variables globales con el mismo valor (-9).

La respuesta correcta es: Falso

. . .

Pregunta 3

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

La única forma de borrar un módulo es con el comando `(reset)`.

Seleccione una:

- a. Falso
- b. Verdadero
- c. No contestar

Página anterior | Siguiente página

. . .

Pregunta 4

Respuesta guardada

Puntúa como 1,00

Marcar pregunta

Podemos incluir un nuevo módulo en la pila de módulos activos usando la orden `focus` en el consecuente de una regla. Por ejemplo,

```
(focus AVANZAR)
```

Seleccione una:

- a. Falso
- b. Verdadero
- c. No contestar

Página anterior | Terminar intento...

. . .