Construcción de SSPP con CLIPS

Susana Fernández Arregui
susana.fernandez@uc3m.es

Raquel Fuentetaja Pizán
raquel.fuentetaja@uc3m.es

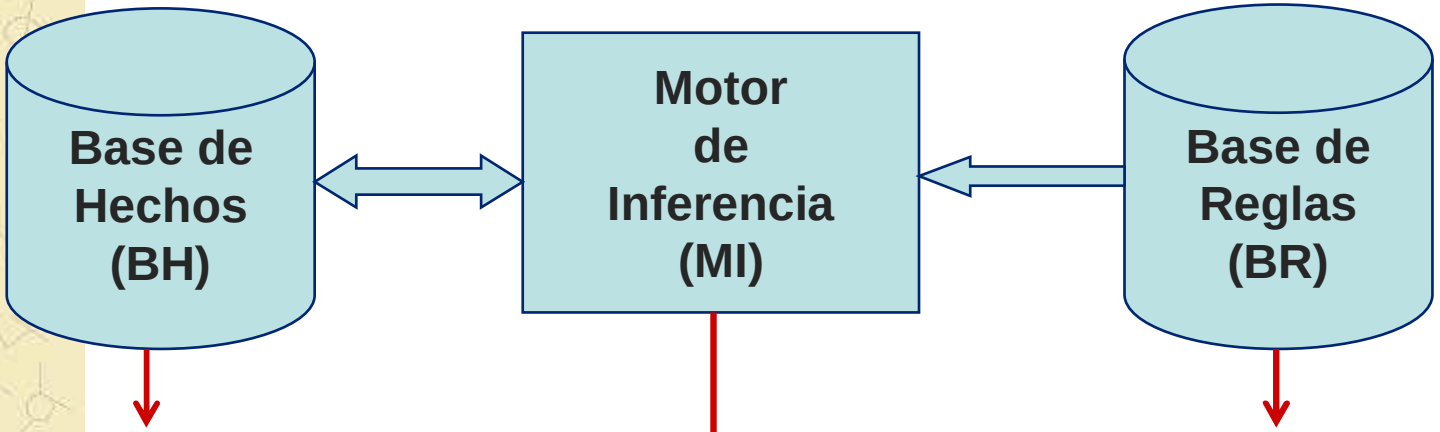
Universidad Carlos III de Madrid

Departamento de Informática

http://galahad.plg.inf.uc3m.es/~docweb/

Índice

Arquitectura de un sistema de producción



También Memoria de Trabajo (MT)
Representa conocimientos del dominio sobre el problema (datos, hechos establecidos y metas a alcanzar)

Representa conocimientos sobre cómo conseguir la solución del problema en forma de reglas de producción

SI <A> ENTONCES <B>

Razona sobre el conocimiento del problema y sobre su solución. Determina cómo se aplican las reglas

Índice

Base de Hechos (BH)

- Representa el **estado actual** del mundo que se modeliza o del problema en curso de solución
  - *Estado inicial: situación inicial*
  - *Estado final: situación final*
  - *Estado intermedio: situación actual o en curso de resolución*
- Parte de la BH es permanente, mientras que otra es temporal (pertenece a la solución del problema en curso)
- Contiene conocimiento declarativo: hechos (facts) e instancias (instances)
- La ejecución de reglas modifica la BH

CLIPS: Base de Hechos

En CLIPS los **hechos** se almacenan en la lista de hechos **fact-list**

- **ORDERED FACTS**
  - Patrones con uno o varios campos
  - El primer campo suele representar una relación entre los restantes
  - La información se codifica por su posición (para acceder a una determinada información es necesario saber qué campo la contiene)

```
(hola)
(alumnos Pepe Juan Luis)
(nombre “Juan”)
(edad 14)
```

CLIPS: Base de Hechos

- **NON-ORDERED FACTS** (**deftemplate facts**)
  - Permiten abstraer la estructura de un hecho asignando un nombre a cada campo
  - El orden de los slots no tiene importancia
  - Permiten definir la información en forma de clases de elementos y atributos de esas clases. El primer nombre del *deftemplate* corresponde con el nombre de la clase

```
(cliente (nombre Juan Pérez)
         (id 33435))
(cliente (id 33435)
           (nombre Juan Pérez))
(clase   (profesor Marta López)
           (estudiantes 30)
           (aula “37A”))
```

CLIPS: Base de Hechos

- Creación de templates

```
(deftemplate <nombre-plantilla>
   [<comentario-opcional>]
    <definición-de-slot1>
    <definición-de-slot2>
    …
    <definición-de-slotN>)
```

```
(deftemplate robot
       (slot bandeja
              (type SYMBOL)
              (allowed-values LLENA VACIA)
              (default VACIA)))
```

- Atributos de los slots
  - De restricción: type, allowed-values, range
  - De valor: default (?NONE, ?DERIVE)

CLIPS: Base de Hechos

En CLIPS las **instancias** se almacenan en la lista de instancias **instance-list**

- Una instancia representa a un individuo concreto perteneciente a una clase
- Las clases permiten construir **marcos** y dotan a CLIPS de las características propias de la programación Orientada a Objetos
  - Abstracción
  - Encapsulación
  - Herencia
  - Polimorfismo

CLIPS: Base de Hechos

- Creación de clases

```
(defclass <nombre-plantilla>
   (is-a <clase-padre>)
   <propiedades de la clase>
    <definición-de-slot1>
    <definición-de-slot2>
    …
    <definición-de-slotN>)
```

```
(defclass ANIMAL (is-a INITIAL-OBJECT)
                 (role abstract)
        (slot edad
            (type INTEGER)
            (create-accessor read-write)))

(defclass ELEFANTE (is-a ANIMAL)
                   (role concrete)
        (slot edad (source-composite)))
```

- Propiedades de la clase
  - (role abstract/concrete)
  - (pattern-match non-reactive/reactive)
- Atributos de los slots
  - Igual que en las templates. Algunos atributos propios
    - (create-accesor read-write) Para poder modificarlo
    - (source-composite) Hereda todas las facetas del mismo slot que el padre

CLIPS: Base de Hechos

- Para no tener que teclear los **hechos iniciales**, podemos introducirlos con la estructura **deffacts**

```
(deffacts hechos-600
        "información del 600"
        (modelo 600)
        (puertas 2))

(deffacts OtrosHechos
        (cliente (nombre Juan Pérez) (id 33435))
        (cliente (nombre Rosa Gómez) (id 33436)))
```

- Para no tener que teclear las instancias iniciales, podemos introducirlas con la estructura **definstances**

```
(definstances instancias-animales
        ([Dumbo] of ELEFANTE (edad 1)))
```

- Estos hechos e instancias se añadirán a la MT al hacer **reset**

CLIPS: Base de Hechos

Etiquetas temporales (time-tag)

- Un *time-tag* es un índice relativo a la creación del hecho (fact-index) o instancia (instance-index)
- Sirven para
  - Identificar de forma única y simplificada cada hecho/instancia (p.e. f-10 es el hecho con etiqueta temporal 10, i-10 es la instancia con etiqueta temporal 10)
  - Saber el tiempo que lleva un hecho/instancia en la memoria de trabajo
- Al primer hecho creado (*initial-fact*) le corresponde la etiqueta temporal f-0
- A la primera instancia creada (*initial-object*) le corresponde la etiqueta temporal i-0
- Nunca disminuye y nunca se reasigna. Para cada nuevo hecho/instancia que se cree o modifique se incrementa en uno

CLIPS: Base de Hechos

ÓRDENES BÁSICAS DE MANEJO DE LA MT

(facts) Lista los hechos de la MT	**(instances)** Lista las instancias de la MT
(assert <hecho>) Añade un hecho a la MT	**(make-instance <instancia>)** Añade una instancia a la MT
(retract <índice-hecho>) Elimina un hecho de la MT	**(unmake-instance <instancia>)** Elimina una instancia de la MT

(clear) Elimina todos los hechos y construcciones de la MT
(reset) Elimina todos los hechos de la MT, elimina las activaciones de la agenda y restaura las condiciones iniciales: ⋆ Añade *initial-fact* e *initial-object* ⋆ Añade el conocimiento inicial definido con *deffacts* y *definstances* ⋆ Añade las variables globales con su valor inicial ⋆ Fija como módulo principal el módulo *main*

Índice

Base de Reglas

- Contiene un conjunto de reglas que representan conocimientos sobre la **solución del problema**. La base de hechos describe el problema, las reglas dan información sobre cómo resolverlo

Si **Condiciones** entonces **Acciones**

Antecedente, LHS — Consecuente, RHS

"El coste del envío se incrementa en 1000 pesetas si se recibe en el mismo día"

```
(defrule coste-del-envio
        (envio ?paquete ?origen ?destino)
        (dia-entrega ?paquete hoy)
        ?coste-inicial<-(coste ?paquete ?precio)
  =>
        (retract ?coste-inicial)
        (assert (coste-final ?paquete (+ ?precio 1000)))
```

CLIPS: Sintaxis de las Reglas

```
(defrule <nombre_regla>
        [<documentacion opcional>]
        (premisa 1)
        (premisa 2)
        ...
        (premisa N)
=>
        (accion 1)
        (accion 2)
        ...
        (accion M))
```

```
(defrule marca-del-600
   “Marca del modelo 600”
    (modelo 600)
=>
    (assert (marca-es SEAT)))
```

CLIPS: Sintaxis de las Reglas

¿Cómo son las premisas del antecedente?

- Las premisas o condiciones de una regla están implícitamente unidas por la conectiva AND
- Una premisa puede ser:
  1. Un patrón
  2. Un test
  3. Una negación

CLIPS: Sintaxis de las Reglas

1. PATRONES

- Un patrón es una consulta a la Memoria de Trabajo para preguntar por la existencia de hechos/instancias en la misma

```
(defrule encontrar-dato
          (dato 1 azul rojo)
 =>)
(defrule encontrar-Roberto
        (persona (nombre Roberto)
                 (edad 20))
 =>)
```

```
(defrule edad-20
        (object (is-a ANIMAL)
                (edad 20))
 =>)
```

- Con un patrón se puede almacenar la dirección de un hecho/instancia en una variable

```
(defrule matrimonio
         ?soltero <- (soltero Juan)
=>
         (retract ?soltero)
         (assert (casado Juan)))
```

CLIPS: Sintaxis de las Reglas

- Dentro de los patrones los campos se pueden sustituir por:
  - **COMODINES**: Cuando no importa el valor de un campo

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

  - **VARIABLES**: Cuando se quiere almacenar el valor de un campo en una variable para usarlo posteriormente

```
(defrule encontrar-dato
      (dato ?x $?y ?z)=>)
```

  - **EL VALOR DE RETORNO DE UNA FUNCIÓN**: Este valor se incorpora directamente en el patrón, en la posición en la que la función es llamada.

```
(defrule ejemplo
        (dato =(+ 4 5))
=>)
```

CLIPS: Sintaxis de las Reglas

2. TEST

- Para comprobar el cumplimiento de una condición
- La función (predefinida/usuario) sobre la que se realiza el test debe devolver un **resultado booleano**:
  - ⋆ Funciones de comparación
    - Numérica: `=, <>, <, >, >=, <=`
    - Cualquier tipo: **`eq, neq`**
  - ⋆ Funciones lógicas: **`or, not, and`**
  - ⋆ Funciones de predicado: **`stringp, numberp, symbolp...`**

```
(defrule ejemplo
        (dato ?x)
        (test (= ?x 3)) =>)
```

CLIPS: Sintaxis de las Reglas

3. NOT

- Las reglas pueden tener como premisa un elemento de condición negado

```
(defrule ejemplo1
        (dato rojo)
        (not (dato rojo ?x ?x)
=>)
```

```
(defrule ejemplo2
        (intevalo-tiempo ?tiempo1 ?tiempo2)
        (not(test (> ?tiempo1 ?tiempo2)))
=>)
```

CLIPS: Sintaxis de las Reglas

¿Cómo son las acciones del consecuente?

- Las acciones de una regla están implícitamente unidas por la conectiva AND
- Las acciones de una regla únicamente se podrán llevar a cabo si se satisfacen todas las premisas del antecedente de la misma
- Una acción puede ser para:
  - La creación de un hecho o instancia en la MT
  - La eliminación de un hecho o instancia de la MT
  - La modificación de un hecho o instancia de la MT
  - Parar la ejecución del sistema de producción
  - Asignar un valor a una variable (bind)
  - Para hacer una operación de entrada/salida
  - Para llamar a una función

CLIPS: Sintaxis de las Reglas

1. assert / make-instance

- **Creación** de un hecho/instancia para incorporarlo en la MT
- Si un hecho idéntico al que se quiere crear ya existe en la MT la acción *assert* no produce ningún efecto

2. retract /unmake-instance

- **Eliminación** de un hecho/instancia de la MT
- Sólo se puede utilizar con argumentos que sean variables o etiquetas temporales

```
(defrule abrir-semáforo
        ?color-disco<-(disco color rojo)
=>
        (retract ?color-disco)
        (assert (disco color verde)))
```

CLIPS: Sintaxis de las Reglas

3. modify / modify-instance

- **Modificación** de un hecho/instancia de la MT
- Es equivalente a hacer un retract y después un assert ⇒ Al hecho modificado se le asigna una **nueva etiqueta temporal**
- Sólo se puede utilizar con non-ordered facts (**template facts**)
- Sólo se puede utilizar con argumentos que sean variables o etiquetas temporales

```
(defrule coste-del-envio
        (envio (paquete ?paquete) (origen ?o) (destino ?d))
        (dia-entrega (paquete ?paquete)(dia hoy))
        ?coste-inicial <- (coste (paquete ?paquete)
                                 (precio ?precio))
=>
        (modify ?coste-inicial (precio (+ ?precio 1000))))
```

CLIPS: Sintaxis de las Reglas

4. halt

- **Parada** del sistema de producción

```
(defrule coste-del-envio
        (envio ?paquete ?origen ?destino)
        (dia-entrega ?paquete hoy)
        ?coste-inicial<-(coste ?paquete ?precio)
=>      (retract ?coste-inicial)
        (assert (coste-final ?paquete (+ ?precio 1000)))
        (halt))
```

5. bind: asignación de valores a variables `(bind ?X (* ?Y 2))`

6. Operación de entrada/salida open, close, printout, read, readline

7. Llamada a función

NO son Reglas

- Una regla **NO** es una estructura **"if-then-else"**

SI premisa1
premisa2
...
ENTONCES
acciones1
EN CASO CONTRARIO
acciones2

¿QUÉ HACER?

- Identificar las condiciones que no se cumplen para la ejecución de acciones2
- Crear dos reglas, una para cada bloque de acciones

NO son Reglas

- **NO** pueden aparecer **OR** en el consecuente de una regla

SI premisa1
premisa2
...
ENTONCES
acciones1 OR acciones2

¿QUÉ HACER?

- Crear dos reglas, una para cada bloque de acciones
- Establecer prioridad entre las reglas

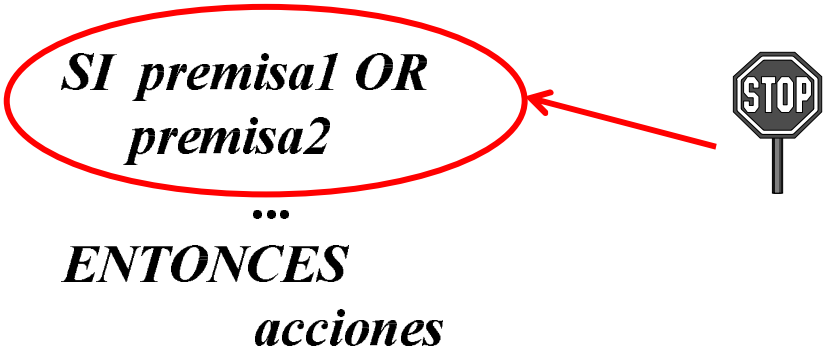
Ri: SI condicion1
condicion2...
ENTONCES acciones1

Rj: SI condicion1
condicion2...
ENTONCES acciones2

Ri más prioritaria que Rj

NO son Reglas

- **NO** deben aparecer **OR** en el antecedente de una regla



¿QUÉ HACER?

- Crear dos reglas, una para cada bloque de premisas
- Establecer prioridades entre las reglas
- Averiguar si faltan condiciones en alguna de las reglas
- Comprobar si las acciones son realmente las mismas

NO son Reglas

- En el consecuente de una regla **NO** hay **elementos de decisión**

SI premisa1 , premisa2 ...
ENTONCES
acciones1
SI premisa4 ...
ENTONCES
acciones2

¿QUÉ HACER?

- Introducir señalizadores que provoquen la ejecución prioritaria de reglas con tales elementos
- No olvidar borrar los señalizadores

Ri: SI premisa1 , premisa2 ...
ENTONCES acciones1
añadir S

Rj: SI S, premisa4 ...
ENTONCES acciones2
borrar S

NO son Reglas

- Desde una regla **NUNCA** se puede **lanzar otra regla**

Ri: SI condicion1 OR
condicion2
...
ENTONCES
Rj

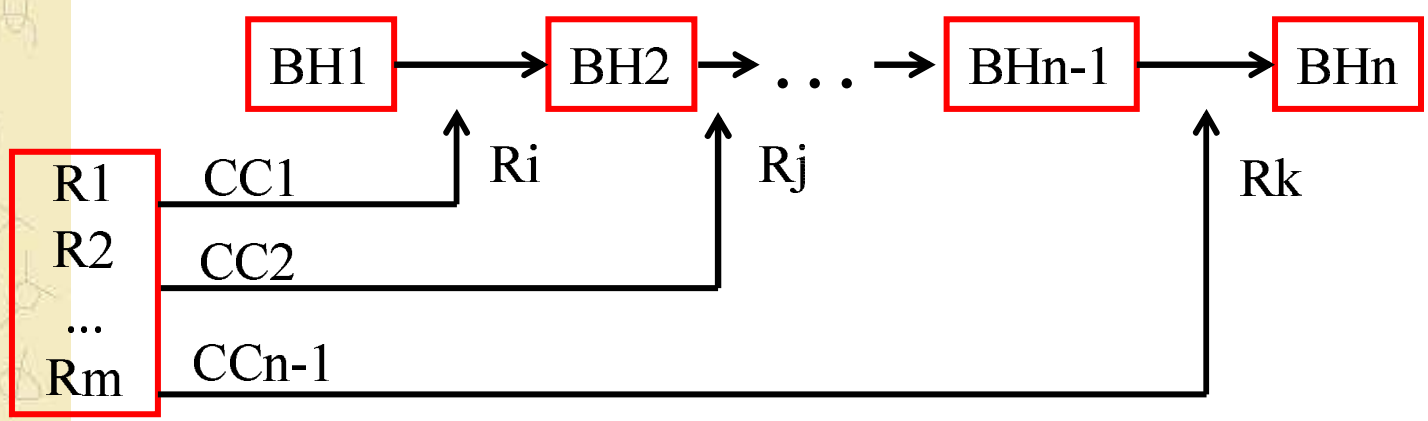
¿QUÉ HACER?

- Introducir señalizadores que provoquen la ejecución de Rj
- No olvidar borrar los señalizadores al ejecutar la regla Rj

Índice

Motor de Inferencia (MI)

- Examina en cada ciclo de inferencia la Base de Hechos y decide qué regla ejecutar



- Carácterísticas
  - ⋆ Causar movimiento
  - ⋆ Ser sistemático
  - ⋆ Ser eficiente

Índice

Ciclo de inferencia: Fases

1. Fase de selección
    a) Restricción
    b) Equiparación
    c) Resolución del Conjunto Conflicto
2. Fase de acción o ejecución

Ciclo de inferencia: Fases

Fase de selección

a) Restricción: Acota el contenido de la BR y la BH según las características del problema a resolver

b) Equiparación: Selección del conjunto de reglas candidatas para dispararse (aquellas cuyo antecedente se satisface)

c) Resolución del Conjunto Conflicto

- **Conjunto Conflicto**: Conjunto de instancias de las posibles reglas ejecutables
- Selección de la instancia de regla que va a ser ejecutada en la fase de acción → Estrategias de Selección

Ciclo de inferencia: Fases

Fase de acción

Ejecución de la instancia seleccionada

- La ejecución de la regla modifica la BH actual dando lugar a una BH nueva al AÑADIR, BORRAR y/o MODIFICAR elementos de la primera
- La BH nueva se tomará como punto de partida en el siguiente ciclo de funcionamiento

Ejemplo de Sistemas de Producción

BASE DE REGLAS

```
(defrule R1
      (animal ?A)
      (esqueleto ?A si)
=> (assert (vertebrado ?A)))


(defrule R2
      (animal ?A)
      (esqueleto ?A no)
=> (assert (invertebrado ?A)))


(defrule R3
      (vertebrado ?A)
      (ladra ?A)
⇒(assert (perro ?A)))
```

BASE DE HECHOS

```
(animal Tucky)
(animal Piolín)
(esqueleto Piolín si)
(esqueleto Tucky si)
(ladra Tucky)
```

Motor de inferencia

Ciclo 1 → R1,?A=Tucky

Ciclo 1 → R1,?A=Piolín

```
(animal Tucky)
(animal Piolín)
(esqueleto Piolín si)
(esqueleto Tucky si)
(ladra Tucky)
(vertebrado Tucky)
```

Ejemplo de Sistemas de Producción

BASE DE REGLAS

```
(defrule R1
      (animal ?A)
      (esqueleto ?A si)
=> (assert (vertebrado ?A)))


(defrule R2
      (animal ?A)
      (esqueleto ?A no)
=> (assert (invertebrado ?A)))


(defrule R3
      (vertebrado ?A)
      (ladra ?A)
⇒(assert (perro ?A)))
```

BASE DE HECHOS

```
(animal Tucky)
(animal Piolín)
(esqueleto Piolín si)
(esqueleto Tucky si)
(ladra Tucky)
(vertebrado Tucky)
```

Motor de inferencia

Ciclo 2

~~R1,?A=Tucky~~

R1,?A=Piolín

R3,?A=Tucky

```
(animal Tucky)
(animal Piolín)
(esqueleto Piolín si)
(esqueleto Tucky si)
(ladra Tucky)
(vertebrado Tucky)
(vertebrado Piolin)
```

Ejemplo de Sistemas de Producción

BASE DE REGLAS

```
(defrule R1
      (animal ?A)
      (esqueleto ?A si)
=> (assert (vertebrado ?A)))


(defrule R2
      (animal ?A)
      (esqueleto ?A no)
=> (assert (invertebrado ?A)))


(defrule R3
      (vertebrado ?A)
      (ladra ?A)
⇒(assert (perro ?A)))
```

BASE DE HECHOS

```
(animal Tucky)
(animal Piolín)
(esqueleto Piolín si)
(esqueleto Tucky si)
(ladra Tucky)
(vertebrado Tucky)
(vertebrado Piolin)
```

Motor de inferencia

Ciclo 3

~~R1,?A=Tucky~~

~~R1,?A=Piolín~~

R3,?A=Tucky

```
(animal Tucky)
(animal Piolín)
(esqueleto Piolín si)
(esqueleto Tucky si)
(ladra Tucky)
(vertebrado Tucky)
(vertebrado Piolin)
(perro Tucky)
```

Índice

Ciclo de Inferencia: Equiparación

Permite elegir aquellas reglas que conducen a la solución del problema al comparar cada una de las condiciones de las reglas con el estado actual de la Memoria de Trabajo

- Equiparación de constantes
- Equiparación de variables

Equiparación de constantes

- Una constante es cualquier secuencia de caracteres no precedidos por el símbolo ?

- Una constante equipara con otra constante igual a ella que

  a) Ocupe la misma posición en un elemento de MT si el patrón que aparece en la regla corresponde a un ordered-fact

  b) Esté asociada mismo campo de la misma clase en un elemento de la MT si el patrón que aparece en la regla corresponde a un non-ordered-fact

Equiparación de constantes (ejemplo)

- Caso a)

BR

```
(defrule R1
      (A B)
=> (assert (C D)))
```

El hecho f-1 equipara con la premisa (A B) de la regla

BH

```
f-1  (A B)
f-2  (C D)
```

El hecho f-2 no equipara

- Caso b)

BR

```
(defrule R1
      (clase1 (nombre1 A))
=> (assert (C D)))
```

El hecho f-1 equipara con la premisa (clase1 (nombre1 A)) de la regla

BH

```
f-1(clase1 (nombre1 A))
f-2(C D)
```

El hecho f-2 no equipara

Equiparación de variables (I)

- Una variable que aparece una sola vez en la regla se equipara con cualquier valor que
  - Ocupe la **misma posición** en un elemento de la MT (**ordered-facts**)
  - Esté asociado **mismo campo de la misma clase** en un elemento de la MT (**non-ordered-facts**)

BR

```
(defrule R1
      (A ?X B)
=>...
```

BH

```
f-1  (A C B)
f-2  (A 3 B)
f-3  (A D C)
```

1) ?X=C
2) ?X=3

Equiparación de variables (II)

- Una variable que aparece dos o más veces en la regla debe equipararse en todas las ocurrencias con el mismo valor

BR

```
(defrule R1
      (animal ?x)
      (piel pelo ?x)

=> (especie mamifero ?x))
```

MT

```
f-1 (animal Tucky)
f-2 (piel pelo Dolly)
f-3 (piel pelo Tucky)
f-4 (animal Dolly)
f-5 (animal Dumbo)
```

1) ?x=Tucky: (animal Tucky) (piel pelo Tucky)
2) ?x=Dolly: (animal Dolly) (piel pelo Dolly)

Dumbo no equipara por no tener (piel pelo Dumbo) en la MT

Equiparación de variables (III)

- Se pueden equiparar variables distintas con el mismo valor

BR

```
(defrule R1
      (A ?X)
      (B ?Y)
=>...
```

1) ?X=C, ?Y=C

BH

```
f-1  (A C)
f-2  (B C)
f-3  (M V)
```

- Se pueden realizar comprobaciones adicionales sobre las variables

```
(defrule R1  (casilla ?i ?j) (test(= ?i 4) (test(> ?j 0)) ...
(defrule R2  (casilla ?i ?j ?color) (test(neq ?color rojo))
             (test(<> ?i ?j)) ...
```

Equiparación de variables (IV)

- Las reglas pueden tener elementos de condición negados (precedidos por *not*)
- La regla equipara si no existen elementos en la MT que hagan cierto el elemento de condición negado
  - ⋆ Hipótesis de mundo cerrado: Todo lo que no está en la MT es falso

BR

```
(defrule R1
      (A ?X)
      (not((B ?X))
=>...
```

BH

```
f-1  (A C)
f-2  (B D)
f-3  (A B)
f-4  (A A)
f-5  (B A)
```

1) ?X=C
2) ?X=B

?X=A no equipara porque en la MT está (B A)

Instancias de las reglas

- Una instancia de una regla es un **par formado por la regla y por los elementos de la MT que hacen cierto el antecedente de la regla**
- Una regla, en un ciclo de funcionamiento puede dar lugar a 0, 1 ó N instancias

BR

```
(defrule R1
      (A ?X)
      (not((B ?X))
=>...
```

BH

```
f-1  (A C)
f-2  (B D)
f-3  (A B)
f-4  (A A)
f-5  (B A)
```

2 instancias de R1

1) ?x=C → R1 (A C) (not (B C))=> ...

2) ?x=B → R1 (A B) (not (B B))=> ...

Conjunto Conflicto={R1(f-1),R1(f-3)}

Ejercicios

1. Suponga un sistema inteligente que atiende varias cesáreas simultáneamente. Se desea modelizar la siguiente heurística:

"Si al hospital llega una madre embarazada con el bebé en posición podálica, el sistema debe recomendar hacer cesárea"

Suponga que las cuatro reglas siguientes modelizan dicha heurística:

```
(defrule R1 (bebe ?b) (posición ?b podalica) => (cesarea ?b))
(defrule R2 (bebe ?b) (posición ?b podalica) => (cesarea ?m))
(defrule R3 (bebe ?b) (posición ?b podalica) (madre ?m) => (cesarea ?m))
(defrule R4 (bebe ?b) (posición ?b podalica) (madre ?m) (madre-de ?m ?b)
        => (cesarea ?m))
```

Identificar las ventajas e inconvenientes de cada modelización

¿Qué sucedería si alguno de los dos bebés de un embarazo gemelar estuviera en posición podálica? ¿Se haría la cesárea a la madre?

Ejercicios

2. Suponga que se han definido los siguientes elementos:

(COLOR ?c ?cc)	Para representar los colores de las fichas con las que juega el ordenador y el contrario
(TURNO ?t)	Para representar que el turno lo tienen las fichas rojas si ?t toma el valor R, o bien B si el turno lo tienen las fichas blancas

En la MT se tienen (COLOR R B) y (TURNO R)

Se desea representar que el ordenador sólo puede mover cuando es su turno ¿Cuál de las siguientes representaciones son correctas?

a) (COLOR ?c ?cc) (TURNO ?t) (test(eq ?c ?t))

b) (COLOR ?c ?cc) (TURNO ?c)

c) (COLOR ?c ?cc) (not(TURNO ?cc))

Índice

Resolución del Conjunto Conflicto

- Selección de la regla que va a ser ejecutada en la fase de acción
- Estrategias de selección
  - Orden de escritura de las reglas en la BR
  - Prioridad (la regla más prioritaria)
  - Especificidad (la regla más específica)
  - Novedad (la regla con elementos de la BH más recientemente añadidos)
  - Utilizar el principio de refracción: No se pueden ejecutar instancias de reglas ya disparadas
  - Arbitrariedad
- Combinación de estrategias

Estrategias de selección

- **Orden de escritura**
  - Se selecciona la primera regla que cuyo antecedente es cierto, siguiendo el orden de escritura en la BR
- **Prioridad**
  - Se selecciona la regla con prioridad más alta
    - La prioridad se establece en función del problema que se modeliza
    - La prioridad la proporciona el experto del dominio

```
(defrule marca-del-600
     “Marca del modelo 600”
     (declare (salience 10))
     (modelo 600)
=>
     (assert (marca-es SEAT)))
```

Estrategias de selección

- **Especificidad**
  - Se selecciona la regla más especifica. La especificidad viene determinada por el número de comparaciones que deben realizarse en el antecedente de la regla
  - Se suma uno a la especificidad por cada:
    - Comparación con una variable previamente asignada (ligada a un valor)
    - Llamada a función que forme parte de una premisa de test o de una asignación del valor de retorno de una función (=), exceptuando llamadas a funciones booleanas y llamadas a funciones anidadas

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

Especificidad = 5

Estrategias de selección

- **Novedad**
  - ⋆ Cada hecho lleva asignado el momento en el que fue creado (time-tag). Se da más prioridad a las instancias de reglas con un hecho más reciente, comparando en orden descendiente

Conjunto Conflicto= {R1(f-4), R2(f-2,f-4)}

- Se ordenan los time-tag en orden decreciente
- Se comparan uno a uno hasta encontrar uno mayor que otro
- En caso de que no haya el mismo número de time-tag se añaden ceros al final

Para R1: 4, 0
Para R2: 4, 2
Por novedad se elige para su ejecución R2

Estrategias de selección

- **Principio de refracción**
  - ⋆ Seleccionar una regla que no se haya instanciado previamente

BR

```
defrule R1 (declare (salience 5)) (A) => (assert (B))
defrule R2 (declare (salience 4)) (B) (not(C)) (not(D)) =>(assert(C))
defrule R3 (declare (salience 3)) (E) => (assert(D))
defrule R4 (declare (salience 2)) (D) => (halt)
defrule R5 (declare (salience 1)) (A) (Aux<-(C)) => (assert(E))
                                                    (retract(Aux))
```

BH

```
f-1  (A)
```

ES

```
Prioridad + P.R.
```

Orden de ejecución: R1,R2,R5,R2,R3,R4

¡ CLIPS siempre utiliza el principio de refracción !

Motor de Inferencias (CLIPS)

- Cuando una regla equipara se dice que está activada
- Las activaciones se mantienen en la **agenda** (Conjunto Conflicto), en que se disponen por orden de prioridad
- Para insertar una activación en la agenda se siguen las estrategias de resolución de conflictos (estrategias de selección)
- La estrategia se selecciona usando el comando **set-strategy**

CLIPS: Estrategias de resolución de conflictos

- **Depth strategy** (estrategia por defecto)
  - Las nuevas activaciones se sitúan por encima de las activaciones con igual prioridad
- **Bread strategy**
  - Las nuevas activaciones se sitúan por debajo de las activaciones con igual prioridad
- **Simplicity strategy**
  - Las nuevas activaciones se sitúan por encima de las activaciones con mayor o igual especificidad
- **Complexity strategy**
  - Las nuevas activaciones se sitúan por debajo de las activaciones con mayor o igual especificidad

CLIPS: Estrategias de resolución de conflictos

- **Random strategy**
  - A cada activación se le asigna un número aleatorio para determinar su orden en la agenda. Siempre se le asigna el mismo número en diferentes ejecuciones
- **LEX strategy**
  - Novedad, especificidad
- **MEA strategy**
  - Se aplica la misma estrategia que LEX pero mirando solo el primer patrón que equipara en la regla. Si coincide se aplica LEX

Conjunto Conflicto= {R1(f-4), R2(f-2,f-4)}

Para R1: 4
Para R2: 2
Por MEA se elige para su ejecución R1

Problemas al añadir y borrar

BR

```
(defrule R1
      (envio ?e ?origen ?destino)
      (dia-entrega ?e hoy)
      (coste ?e ?coste)
=>
      (assert (coste ?e (+ ?coste 1000))))
```

Motor de inferencia

Ciclo1

?e=E1
?origen=Madrid
?destino=Barcelona
?coste=700

BH

```
f-1  (envio E1 Madrid Barcelona)
f-2  (dia-entrega E1 hoy)
f-3  (coste E1 700)
```

Conjunto Conflicto={R1(f-1,f-2,f-3)}

BH

```
f-1  (envio E1 Madrid Barcelona)
f-2  (dia-entrega E1 hoy)
f-3  (coste E1 700)
f-4  (coste E1 1700)
```

ERROR: El coste del envío es único
SOLUCIÓN: Modificar la regla

Problemas al añadir y borrar

BR

```
(defrule R1
      (envio ?e ?origen ?destino)
      (dia-entrega ?e hoy)
      coste-inicial <- (coste ?e ?coste)
=>
      (assert (coste ?e (+ ?coste 1000)
      (retract ?coste-inicial))
```

BH

```
f-1  (envio E1 Madrid Barcelona)
f-2  (dia-entrega E1 hoy)
f-3  (coste E1 700)
```

Motor de inferencia

Ciclo1

?e=E1
?origen=Madrid
?destino=Barcelona
?coste=700

Conjunto Conflicto={R1(f-1,f-2,f-3)}

BH

```
f-1  (envio E1 Madrid Barcelona)
f-2  (dia-entrega E1 hoy)
f-3  (coste E1 700)
f-4  (coste E1 1700)
```

PROBLEMA: Bucle infinito
SOLUCIÓN: introducir señalizadores, utilizar un nuevo hecho (coste-final)

Índice

Estrategias de inferencia

- **Encadenamiento hacia delante** (dirigido por el antecedente o forward chaining)
  - Se parte de unos hechos particulares (el contenido de la BH)
  - Se buscan reglas cuyo antecedente esté satisfecho por esos hechos (BH)
  - Se modifica la BH ejecutando las acciones del consecuente de alguna de las instancias de reglas
- **Encadenamiento hacia atrás** (dirigido por el consecuente o backward chaining)
  - Se parte de un conjunto de metas u objetivos
  - Se buscan reglas cuya instanciación implica la consecución de una meta. **La equiparación se realiza sobre el consecuente**
  - Si el antecedente equipara con la BH la meta se puede concluir
  - Si el antecedente no equipara con la BH se aplica recursivamente el procedimiento utilizando como submetas las expresadas por el antecedente