Bloque 1–
Representación del conocimiento y búsqueda

Tema 6:
Inferencia en SBR:
encadenamiento y control.

Bloque 1, Tema 2- Indice

1. Motor de inferencia
2. Estrategias de resolución de conflictos.
3. Ejemplos.
4. Ejercicio inferencia resuelto.
5. Ejercicios propuestos.
6. Anexo: sintáxis BNF de las reglas en CLIPS

Bibliografía

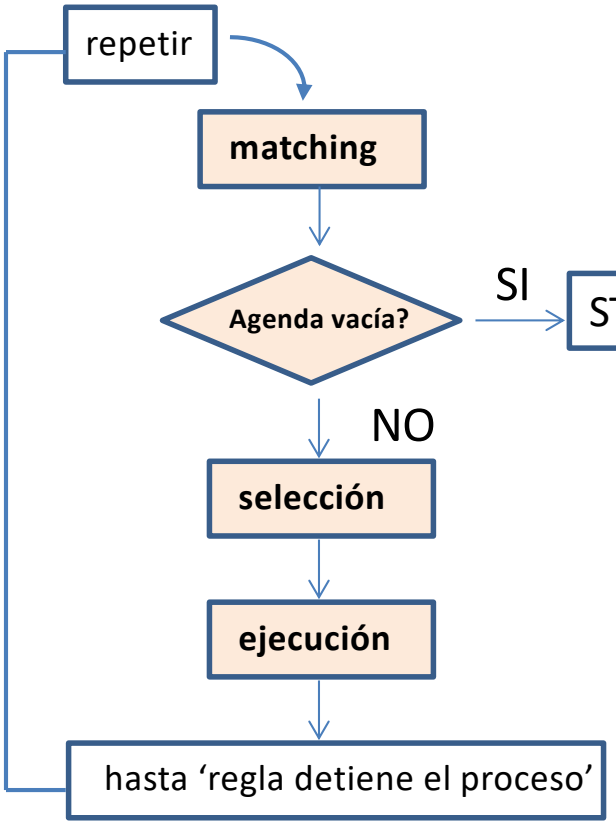
- Capítulo 3: *Sistemas Basados en Reglas*. Inteligencia Artificial. Técnicas, métodos y aplicaciones. McGraw Hill, 2008.
- Charles Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", *Artificial Intelligence*, 19, pp 17–37, 1982.
- CLIPS User's Guide.
- CLIPS Basic Programming Guide.

1. Motor de inferencia

La programación lógica sigue una semántica declarativa. En cambio, un SBR tiene una semántica procedural. Esta diferencia se debe a la naturaleza procedural de la RHS de las reglas en un SBR.

CLIPS utiliza un motor de inferencia hacia delante basado en el algoritmo de matching Rete (motor de inferencia basado en Rete)

El mecanismo de control que siguen los motores de inferencia hacia delante se denomina **ciclo reconocimiento-acción** (también ciclo matching-selección-acción):



Matching de las reglas: genera un Conjunto conflicto (o Agenda) con todas las reglas aplicables o instancias de reglas encontradas

Si la Agenda está vacía, el proceso finaliza. En caso contrario, se selecciona una instancia del Conjunto Conflicto o Agenda de acuerdo a un criterio predeterminado (Estrategia de Resolución de Conflictos).

Se ejecuta la RHS de la instancia seleccionada, actualizando la BH y/o ejecutando las acciones externas

1. Motor de inferencia

BH = base de hechos; BR= base de reglas; *CC*= conjunto conflicto; InstRule=instancia de regla;

```
BH=hechos inciales
CC=Matching(BH,BR)
while objetivo ⊄ BH y CC ≠ ∅
    InstRule=Seleccionar(CC)
    BH=BH \ hechos eliminados por InstRule      ;; Ejecución de la RHS (1)
    BH=BH ∪ hechos añadidos por InstRule        ;; Ejecución de la RHS (2)
    CC=CC \ instancias de reglas eliminadas resultado de hechos eliminados (3)
    CC=CC ∪ Matching(BH,BR)                                          (4)
end_while
if objetivo ⊆ BH        ;; disparo de una regla que encuentra el objetivo y para el MI
    then EXITO
    else FALLO
```

(1) (2): ejecución de la RHS de InstRule ; ejecutar los comandos 'retract' y 'assert' y actualizar la BH en consonancia

(3) Actualizar el CC las instancias que dependen de los hechos eliminados (retract)

(4) Activación de la fase de matching con los nuevos hechos generados (assert)

1. Motor de inferencia: ejemplo 1

```
Ordenar una lista

(deffacts data
(lista 4 5 3 46 12 10)
)


defrule ordenar

?a <- (lista $?x ?y ?z $?w)
(test (< ?z ?y))

=>

(retract ?a)
(assert (lista $?x ?z ?y $?w)))
```

2. Estrategias de resolución de conflictos

El **Conjunto Conflicto** o **Agenda**: colección de instancias de reglas o activaciones resultante del proceso de matching entre las reglas de la BR y los hechos de la BH. En la Agenda puede haber cero o más instancias de reglas.

Estrategia de resolución de conflictos: criterio para seleccionar la activación de la Agenda

Refracción

Una regla solo se puede instanciar una vez con los mismos hechos (hechos con los mismos índices) y mismas instanciaciones de variables. Cuando la BH se modifica, todas las reglas se pueden volver a utilizar de nuevo siempre que se haya producido al menos un nuevo hecho que genere una nueva activación de la regla.

CLIPS aplica refracción, no permitiendo que una regla se dispare dos veces con los mismos datos, evitando así que se disparen repetidamente reglas con los mismos hechos (bucle infinito).

Este comportamiento se complementa con la opción por defecto de CLIPS de no permitir duplicidad de hechos. CLIPS no acepta hechos duplicados, es decir, hechos idénticos pero con diferentes índices (este comportamiento se puede, no obstante, anular seleccionado otra opción).

2. Estrategias de resolución de conflictos: ejemplo refracción y hechos duplicados

```
(defrule R1
      (lista $?x ?y ?z $?w)
      (test (evenp ?z))
   =>
      (assert (lista $?x ?z ?y $?w)))
```

BH= { f-1: (lista 12 5 24 7)}

Matching

~~R1: f-1, $?x=(12), ?y=5, ?z=24, $?w=(7)~~

Selección Ejecución →

BH= { f-1: (lista 12 5 24 7)
f-2: (lista 12 24 5 7)}

Matching

~~R1: f-2, $?x=(), ?y=12, ?z=24, $?w=(5 7)~~

Selección Ejecución →

BH= { f-1: (lista 12 5 24 7)
f-2: (lista 12 24 5 7)
f-3: (lista 24 12 5 7)}

La activación R1: f-1, $?x=(12), ?y=5, ?z=24, $?w=(7) no se vuelve a insertar

Matching

~~R1: f-3, $?x=(), ?y=24, ?z=12, $?w=(5 7)~~

Selección Ejecución →

BH= { f-1: (lista 12 5 24 7)
f-2: (lista 12 24 5 7)
f-3: (lista 24 12 5 7)}

CLIPS no vuelve a insertar el hecho (lista 12 24 5 7)

2. Estrategias de resolución de conflictos: agenda

Cuando hay más de una instancia en la Agenda, la estrategia de resolución de conflictos decide la activación a escoger, de modo que **la instancia que aparece en el top de la agenda es siempre la instancia que se selecciona para su ejecución.**

Prioridades explícitas

Valor numérico del salience de las reglas

En CLIPS, el rango del salience va desde -10,000 hasta 10,000. Cuanto mayor sea el valor, más alta es la prioridad de la regla. Si no se especifica salience en la regla entonces su prioridad es 0.

```
(defrule <rule name> ["comment"]
    (declare (salience <integer>))
        <conditional-element>*      ; left-hand side (LHS)  de la regla
                                     ; condiciones a satisfacer
=>

....
```

2. Estrategias de resolución de conflictos: agenda

Ejemplo: queremos seleccionar 5 personas para jugar al baloncesto, preferiblemente gente con una altura mayor de 2m; sino, escoger gente por debajo de los 2m.

```
(defrule over-2m
    (declare (salience 30))
?f1 <-  (height ?per ?tall)
         (test (>= ?tall 2))
 ?f2 <- (count ?numper)
 =>
    (retract ?f1 ?f2)
    (assert (count (+ ?numper 1))))

(defrule below-2m
     (declare (salience 10))
 ?f1<-  (height ?per ?tall)
 ?f2 <-  (count ?numper)
 =>
   (retract ?f1 ?f2)
   (assert (count (+ ?numper 1))))
```

```
(defrule final-1
   (declare (salience 100))
      (count 5)
    =>
     (halt)
     (printout t "We already have 5 people to play basket " crlf))

 (defrule final-2
     =>
       (halt)
       (printout t "We were not able to find 5 people to play basket ", crlf))

  (deffacts data
     (height John 2.02) (height Peter 1.92)(height Terry 1.86)
     (height Lucas 2.01)(height Nick 2.05)(height Joshua 1.94)
     (count 0))
```

2. Estrategias de resolución de conflictos: agenda

Profundidad

Las nuevas activaciones se manejan como una cola de prioridades. Los empates se resuelven mediante una estrategia LIFO (más prioridad para los hechos e instancias más recientes).

Anchura

Las nuevas activaciones se manejan como una cola de prioridades. Los empates se resuelven mediante una estrategia FIFO (más prioridad para los hechos e instancias más antiguos).

Aleatoria

A cada activación se le asigna un número aleatorio que se usa para determinar la posición de la activación entre aquellas activaciones que tienen el mismo *salience*. CLIPS permite seleccionar **Random** como estrategia de resolución de conflictos.

3. Ejemplos

En las siguientes transparencias se muestran 4 trazas de un SBR y se analiza el funcionamiento del Motor de Inferencia de CLIPS.

Para cada ejemplo se define: a) la BH inicial; b) las reglas que forman la BR; y c) la estrategia de resolución de conflictos

El objetivo es analizar la sucesiva aplicación del ciclo matching-selección-ejecución hasta que el problema finaliza (en estos ejemplos el problema finaliza cuando la agenda se queda vacía).

3. Ejemplo 1

Asumiendo que la estrategia de la Agenda es **ANCHURA**, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales dada.

BH={(list 3 2 7 5)}

```
(defrule R1
    ?f <- (list $?x ?y ?z $?w)
    (test (< ?z ?y))
=>
    (assert (list $?x ?z ?y $?w)))
```

BH (hechos)	Agenda (instancias de reglas)	
f-1: (list 3 2 7 5)	R1: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=1}	selección (1)
	R1: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=1}	selección (2)
f-2: (list 3 2 5 7)	R1: f-2 {$?x=(),?y=3, ?z=2, $?w=(5 7), ?f=2}	selección (3)
f-3: (list 2 3 7 5)	R1: f-3 {$?x=(2 3),?y=7, ?z=5, $?w=(), ?f=3}	selección (4)
f-4: (list 2 3 5 7)	no hay matching	
no hay nuevos hechos (hecho duplicado)		

BH final={(list 3 2 7 5) (list 3 2 5 7) (list 2 3 7 5)(list 2 3 5 7)}

3. Ejemplo 2

Asumiendo que la estrategia de la Agenda es **ANCHURA**, realiza una traza del siguiente SBR y muestra el contenido final de la BH. Este SBR ordena una lista de números naturales dada.

BH={(list 3 2 7 5)}

```
(defrule R1
    ?f <- (list $?x ?y ?z $?w)
    (test (< ?z ?y))
=>
    (retract ?f)
    (assert (list $?x ?z ?y $?w)))
```

	BH (hechos)	Agenda (instancias de reglas)	
eliminar (1) →	f-1: (list 3 2 7 5)	R1: f-1 {$?x=(3 2),?y=7, ?z=5, $?w=(), ?f=1}	← selección (1)
		R1: f-1 {$?x=(),?y=3, ?z=2, $?w=(7 5), ?f=1}	← eliminar(1)
eliminar (2) →	f-2: (list 3 2 5 7)	R1: f-2 {$?x=(),?y=3, ?z=2, $?w=(5 7), ?f=2}	← selección (2)
	f-3: (list 2 3 5 7)	no hay matching	

BH final={(list 2 3 5 7)}

3. Ejemplo 3

Asumiendo que la estrategia de la Agenda es **ANCHURA** (mayor prioridad para las instancias y los hechos más antiguos comenzando por la regla 'move'), realiza una traza del siguiente SBR y muestra el contenido final de la BH.

BH={(list a b c d e) (move-to-front c)}

```
(defrule move
   ?m<- (move-to-front ?who)
    ?l <- (list $?front ?who $?rear)
=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
    ?ch <- (change-list yes)
           (list $?list)
=>
    (retract ?ch)
    (printout t "List is " $?list crlf))
```

BH (hechos)	Agenda (instancias de reglas)
f-1: (list a b c d e) f-2: (move-to-front c)	move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e) ?m=2, ?l=1} ← selección (1)
f-3: (list c a b d e) f-4: (change-list yes) ← eliminar (3)	move: f-2,f-3 {?who=c, $?front=(), $?rear=(a b d e) ?m=2, ?l=3} ← selección (2) print: f-4,f-1 {$?list=(a b c d e), ?ch=4} ← selección (3) print: f-4,f-3 {$?list=(c a b d e), ?ch=4} ← eliminar (3)
(hecho duplicado)	

```
CLIPS> (run)
List is (a b c d e)
```

BH final={(list a b c d e)(move-to-front c)(list c a b d e)}

3. Ejemplo 4

Asumiendo que la estrategia de la Agenda es **PROFUNDIDAD** (mayor prioridad para las instancias y los hechos más recientes comenzando por la regla 'move'), realiza una traza del siguiente SBR y muestra el contenido final de la BH.

BH={(list a b c d e) (move-to-front c)}

```
(defrule move
   ?m<- (move-to-front ?who)
    ?l <- (list $?front ?who $?rear)
=>
   (assert (list ?who $?front $?rear))
   (assert (change-list yes)))
```

```
(defrule print
    ?ch <- (change-list yes)
           (list $?list)
=>
    (retract ?ch)
    (printout t "List is " $?list crlf))
```

BH (hechos)	Agenda (instancias de reglas)
f-1: (list a b c d e) f-2: (move-to-front c)	move: f-2,f-1 {?who=c, $?front=(a b), $?rear=(d e) ?m=2, ?l=1}
f-3: (list c a b d e) f-4: (change-list yes)	print: f-4,f-3 {$?list=(c a b d e), ?ch=4} print: f-4,f-1 {$?list=(a b c d e), ?ch=4} move: f-2,f-3 {?who=c, $?front=(), $?rear=(a b d e) ?m=2, ?l=3}
(hecho duplicado)	

selección (1)

eliminar (2)

selección (2)

eliminar (2)

selección (3)

```
CLIPS> (run)
List is (c a b d e)
```

3. Ejemplo 4 (continuación)

BH (hechos)	Agenda (instancias de reglas)
f-1: (list a b c d e) f-2: (move-to-front c)	
f-3: (list c a b d e)	
f-5: (change-list yes)	print: f-5,f-3 {$?list=(c a b d e), ?ch=5} print: f-5,f-1 {$?list=(a b c d e), ?ch=5}

eliminar (4) →

← selección (4)

← eliminar (4)

CLIPS> (run)
List is (c a b d e)

BH final={(list a b c d e)(move-to-front c)(list c a b d e)}

3. Ejemplos: conclusiones

La elección de la estrategia de resolución de conflictos, especialmente cuando existen eliminaciones de hechos en la RHS de las reglas, puede conducir a ejecuciones distintas y marcar así una diferencia en la evaluación del SBR.

Debido a los efectos de la estrategia de resolución de conflictos, las reglas interaccionan entre sí de un modo u otro y el orden de disparo de las reglas marca la diferencia.

Para entender el funcionamiento de un SBR es necesario no solo entender el significado semántico de las reglas sino considerar también el funcionamiento del MI y tener en cuenta las condiciones bajo las cuales se disparan las reglas.

No es posible entender el significado de una regla como una unidad independiente dentro de un SBR. Hay que considerar el funcionamiento de la estrategia de resolución de conflictos y las relaciones que surgen entre las reglas.

4. Ejercicio inferencia resuelto

Dada una lista de números, queremos diseñar un SBR que encuentre todos los números de la lista que sean múltiplos de dos números primos. Los hechos que se proporcionan son la lista de números y una lista con 6 números primos:

```
(deffacts data
   (lista-numeros 78 45 961 23 362 1872 3041 28)
   (lista-primos 2 3 5 7 11 13)
)

(defrule uno
    (lista-numeros $?x1 ?num $?y1)
    (lista-primos $?x2 ?primo $?y2)
    (test (= (mod ?num ?primo) 0))
 =>
    (assert (multiplo ?primo ?num)))


(defrule dos
   (declare (salience 100))
    (multiplo ?primo1 ?num)
    (multiplo ?primo2 ?num)
    (test (neq ?primo1 ?primo2))
    ?f <- (lista-numeros $?x ?num $?y)
 =>
    (retract ?f)
    (assert (lista-numeros $?x $?y))
    (printout t "El número " ?num " es múltiplo de los primos " ?primo1 " y " ?primo2
crlf))
```

5. Ejercicios propuestos

Ejercicio 1

Dado un SBR, con una única regla:

```
(defrule regla-1
?f <- (lista ?x $?y ?x $?y ?x)
=>
(retract ?f)
(assert (lista $?y)))
```

, y la BH inicial {(lista a b c b c b c b c b a b c b c b c b c b a)}, ¿cuál será el estado final de la Base de Hechos?

A. {(lista a)}
B. {(lista a) (lista)}
C. {(lista b a b)}
D. {(lista c b c)}

5. Ejercicios propuestos

Ejercicio 2

Sea el hecho:

(escuela clase 1 niños 15 niñas 18 clase 2 niños 21 niñas 14 clase 3 niños 16 niñas 17)

, donde el número que aparece después del símbolo 'clase' indica el identificador de dicha clase, y los valores numéricos que aparecen después de los símbolos 'niños' o 'niñas' indican el número de niños o niñas de la clase correspondiente. Indica el patrón adecuado para obtener únicamente el identificador de una clase cualquiera y el número de niñas de dicha clase:

A. (escuela $? clase ?c $? niñas ?na $?)
B. (escuela $? clase ?c niños ? niñas ?na $?)
C. (escuela $? clase ? niños $? niñas ?na $?)
D. (escuela clase ?c niños ? niñas ?na)

5. Ejercicios propuestos

Ejercicio 3

Sea un SBR, con una única regla:

```
(defrule R1
?f <- (lista ?x $?y ?x $?z)
=>
(retract ?f)
(assert (lista $?y ?x $?z))
(printout t “La lista se ha modificado “ crlf))
```

, y la BH inicial {(lista a b a b a)}. Tras ejecutar el SBR, ¿cuántas veces se habrá mostrado en pantalla el mensaje “La lista se ha modificado “?

A. 1
B. 2
C. 3
D. 4

5. Ejercicios propuestos

Ejercicio 4

Dado el siguiente SBR, ¿cuántas reglas se insertarán en la agenda en el primer ciclo de inferencia?

```
(defrule R1
   (lista $?x1 ?y $?x2 ?y $?x3)
=>
   (assert (lista $?x1 ?y $?x3)))


(deffacts inicio
              (lista 2 3 1 2 3 2 1))
```

A. 4
B. 5
C. Ninguna
D. 3

5. Ejercicios propuestos

Ejercicio 5

Dado el siguiente SBR, indica cuál de las siguientes respuestas es CORRECTA:

```
(defrule R1
  (declare (salience 100))
  ?f <- (lista $?x ?y)
  (test (> ?y 5))
=>
  (retract ?f)
  (assert (lista $?x)))

(defrule R2
  (declare (salience 200))
  ?f <- (lista ?y $?x)
  (test (> ?y 5))
=>
  (retract ?f)
  (assert (lista $?x)))

(deffacts inicio
    (lista 3 7 1 5 9))
```

A. Sólo en el caso de que la estrategia de la agenda sea anchura, se ejecutará en primer lugar una instancia de R1
B. Sólo en el caso de que la estrategia de la agenda sea profundidad, se ejecutará en primer lugar una instancia de R2
C. Se ejecutará una instancia de R1 en primer lugar en cualquier caso
D. Se ejecutará una instancia de R2 en primer lugar en cualquier caso

6. Anexo: sintáxis BNF de las reglas en CLIPS

(defrule <rule-name> [<comment>]
    [<declaration>]
    <conditional-element>*
=>
   <action>*)

6. Anexo: sintáxis BNF de las reglas en CLIPS (LHS)

<declaration> ::= (declare (salience <integer-expression>))

<conditional-element>::= (and <CE>) | (or <CE>) | (not <CE>)| (test <function-call>) |
<ordered-pattern> | <assigned-pattern>

<assigned-pattern> ::= <single-vble> ← <ordered-pattern>

<ordered-pattern> ::= (<symbol> <constraint>*)

<constraint> ::= ? | $? | <constant> | <variable>

<constant> ::= <symbol> | <string> | <integer> | <float> | <instance-name>

<variable> ::= <single-vble> | <multi-vble>

<single-vble> ::= ?<symbol>

<multi-vble> ::= $?<symbol>

<CE> ::= <ordered-pattern> | (and <CE>) | (or <CE>) | (not <CE>)

<function-call> ::= (<function-name> <expression>*)

<function-name> ::= = | > | >= | < | <= | <> | eq | neq | <member> ;; just a few examples

<expression> ::= <constant> | <variable> | <function-call>

<term> ::= <single-term> | <multi-term>

<single-term>::= <constant> | <single-vble>

<multi-term>::= <multi-expression> | <multi-vble>

<member> ::= (member <single-term> <multi-term>)

6. Anexo: sintáxis BNF de las reglas en CLIPS (RHS)

<action> ::= (assert <ordered-pattern>) | (retract <term-retract>*) |
(bind <variable> <term-bind>) | <action-multi-value>

<term-retract> ::= <integer> | <single-vble>
<term-bind> ::= <constant> | <variable> | <function-call>
<action-multi-value> ::= (printout …) | (read ….) | (readline ….)