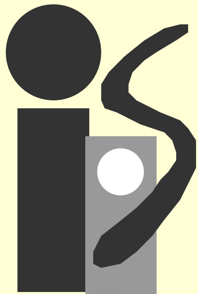
Eficiencia en Sistemas de Reconocimiento de Patrones



J.A. Bañares Bañares

Departamento de Informática e Ingeniería de Sistemas
C.P.S. Universidad de Zaragoza

Copyright © 2000 José Angel Bañares

Última revisión: Octubre, 2002

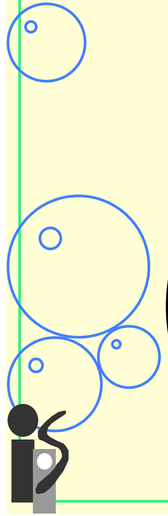
© José Angel Bañares

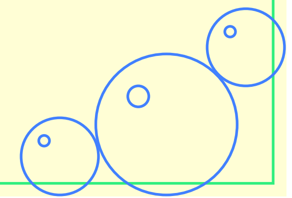
Eficiencia en Sistemas de Reconocimiento de Patrones

- **Índice:**
  - Introducción
  - Proceso de reconocimiento de patrones
  - Registro del estado entre operaciones
  - Algoritmo de RETE
  - Algoritmo de TREAT
  - ¿Cómo escribir programas eficientes?
  - Otras aproximaciones
  - Otras aplicaciones: Interpretación de RdP

© José Angel Bañares

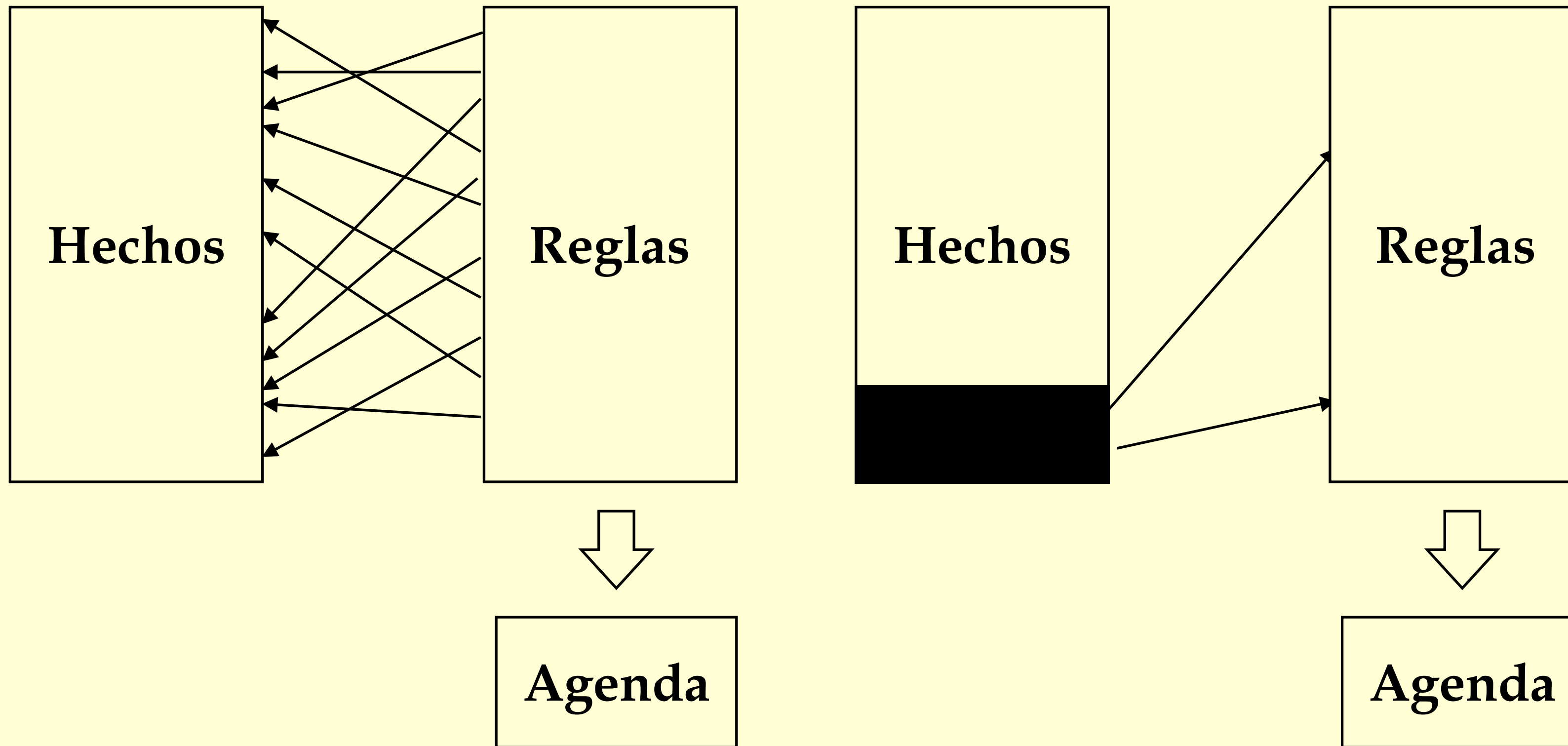
Introducción



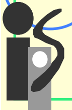


© José Angel Bañares

Introducción



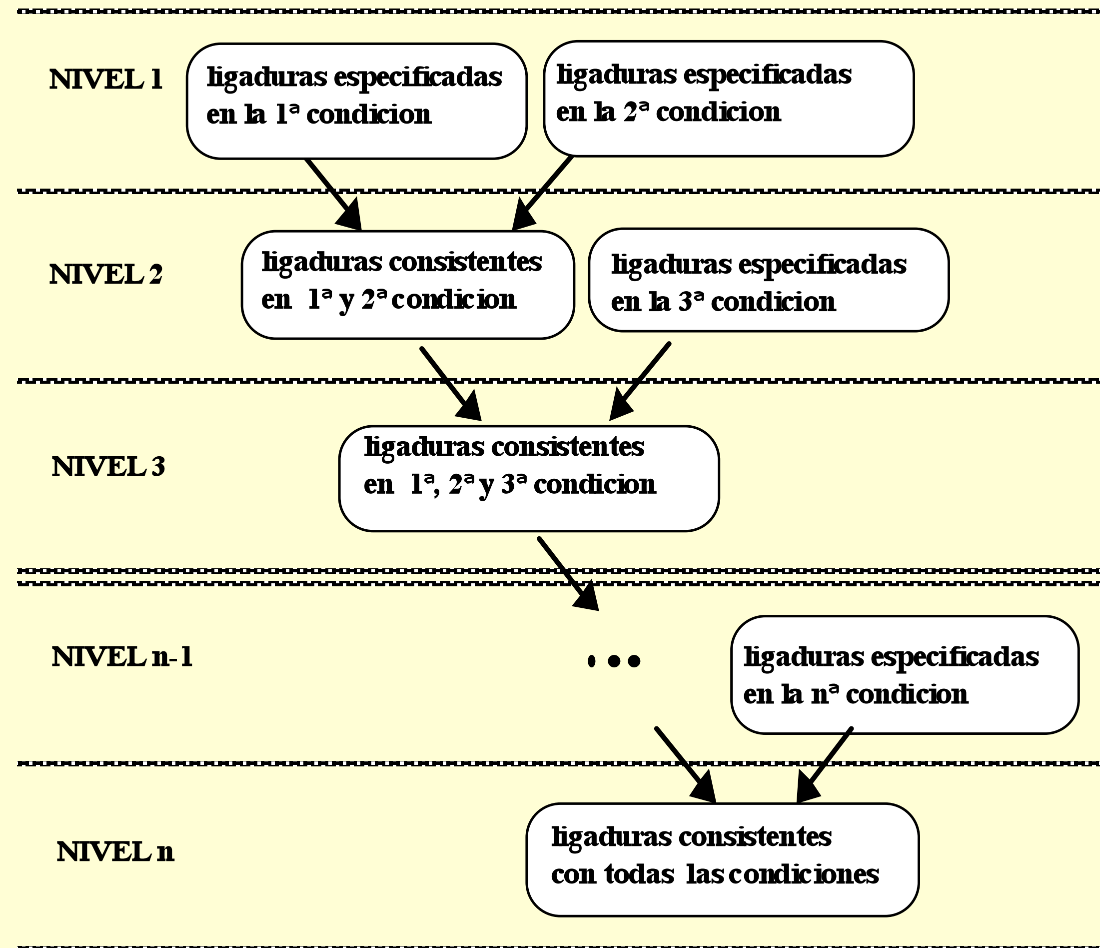
Redundancia Temporal

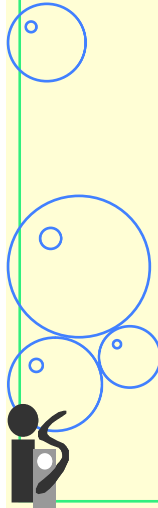


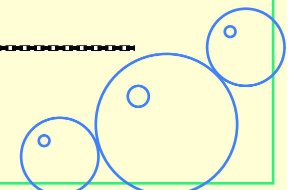
© José Angel Bañares

Proceso de correspondencia de patrones

```
(defrule  nombre-regla
 (primera-condición)
 (segunda-condición)
  ...
 (n-ésima condición)
 =>
  ...)
```





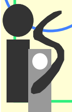


© José Angel Bañares

Registro del estado entre operaciones

- **Tipos de información a retener :**
  McDermott, Newell y Moore 1978

  - Número de condiciones satisfechas
    Condition membership
  - Mantenimiento de la memoria (α-memorias)
    Memory support
  - Relación entre condiciones (β -memorias)
    Condition relationship
  - Mantenimiento del conjunto conflicto
    Conflict set support



© José Angel Bañares

Algoritmo de RETE

- **Red de RETE y algoritmo de propagación**

  Forgy 1982

  – Aprovecha la **redundancia temporal**
    » **α**-memorias: Árbol de discriminación (*test tree*)
    » **β**-memorias: Árbol de consistencia (*join tree)*
    » Agenda (*Conflict set*)
  – Aprovecha la **similitud estructural**
    » Si se repite la misma restricción en distintas reglas se aprovecha la misma rama.

© José Angel Bañares

Red RETE y algoritmo de propagación

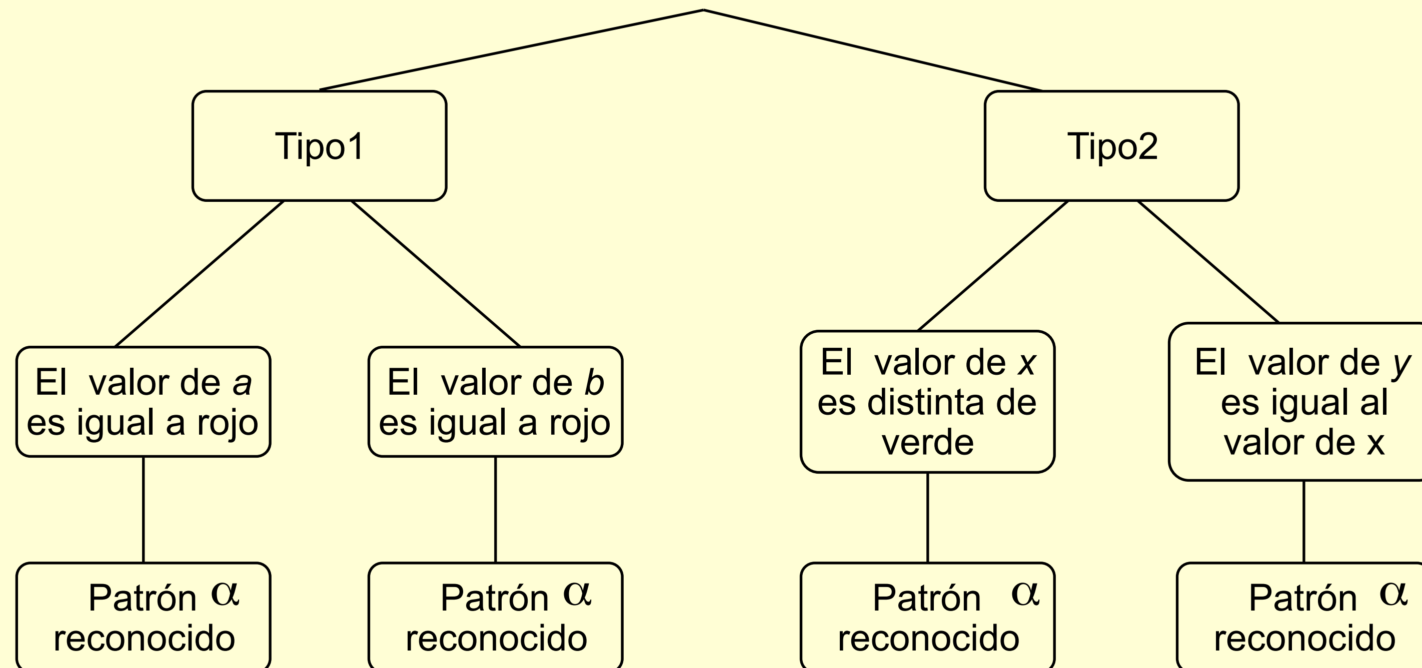
- **Red Rete**
  - Las reglas se compilan bajo la forma de dos tipos de árbol :
    - » Árbol de discriminación : Filtrado y propagación de los nuevos hechos en memoria de trabajo
      - Cada hoja contiene una premisa y cada nodo una comprobación (*test*)
    - » Árbol de consistencia : Verifica las correspondencias entre variable
      - Cada nodo reagrupa 2 a 3 las premisas de una misma regla.

© José Angel Bañares

Árbol de discriminación (test tree)

```
(defrule  Regla-1
   (tipo1 (a rojo))
   (tipo-2 (x ?x) (y ?x))
  => ...)
```

```
(defrule  Regla-2
   (tipo1 (a ?x) (b rojo))
   (tipo-2 (x ~verde) (y ?x))
   (tipo-2 (x ?x) (y ?x))
  => ...)
```



© José Angel Bañares

Árbol de consistencia (Join tree)

Memoria de Trabajo

Tipo1

Tipo2

El valor de *a* es igual a rojo

Patrón α reconocido

El valor de *b* es igual a rojo

Patrón α reconocido

El valor de *x* es distinta de verde

Patrón α reconocido

El valor de *y* es igual al valor de x

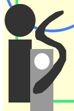
Patrón α reconocido

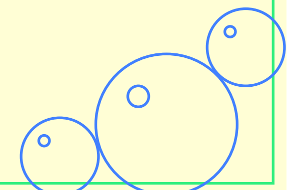
β El valor del atributo *a* del hecho ligado en el primer patrón, **es igual al** valor del atributo *y* del hecho ligado en el segundo patrón

β El valor del atributo *x* del hecho ligado en el tercer patrón, **es igual al** valor del atributo *y* del hecho ligado en el segundo patrón

Regla-2 ACTIVA

```
(defrule  Regla-2
   (tipo1 (a ?x) (b rojo))
   (tipo-2 (x ~verde) (y ?x))
   (tipo-2 (x ?x) (y ?x))
=>)
```



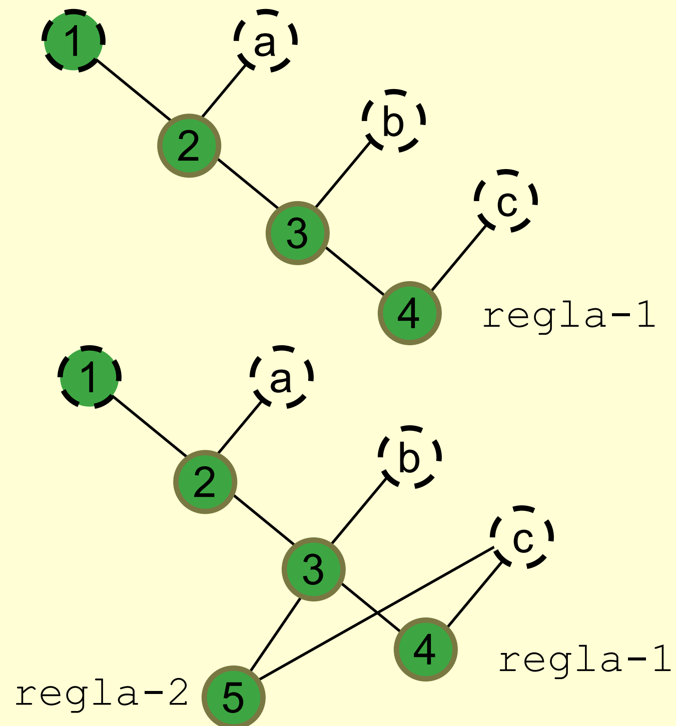


© José Angel Bañares

Compartiendo nodos

- **Similitud estructural**

```
(defrule regla-1
   (tipo-1 (a ?x) (b rojo))    (1)
   (tipo-2 (x ~verde) (y ?x))  (a)
   (tipo-2 (x ?x) (y ?x))      (b)
   (tipo-3 (q ?z))             (c)
   =>)
(defrule regla-2
   (tipo-1 (a ?y) (b rojo))    (1)
   (tipo-2 (x ~verde) (y ?y))  (a)
   (tipo-2 (x ?y) (y ?y))      (b)
   (tipo-3 (q ?y))             (c)
   =>)
```

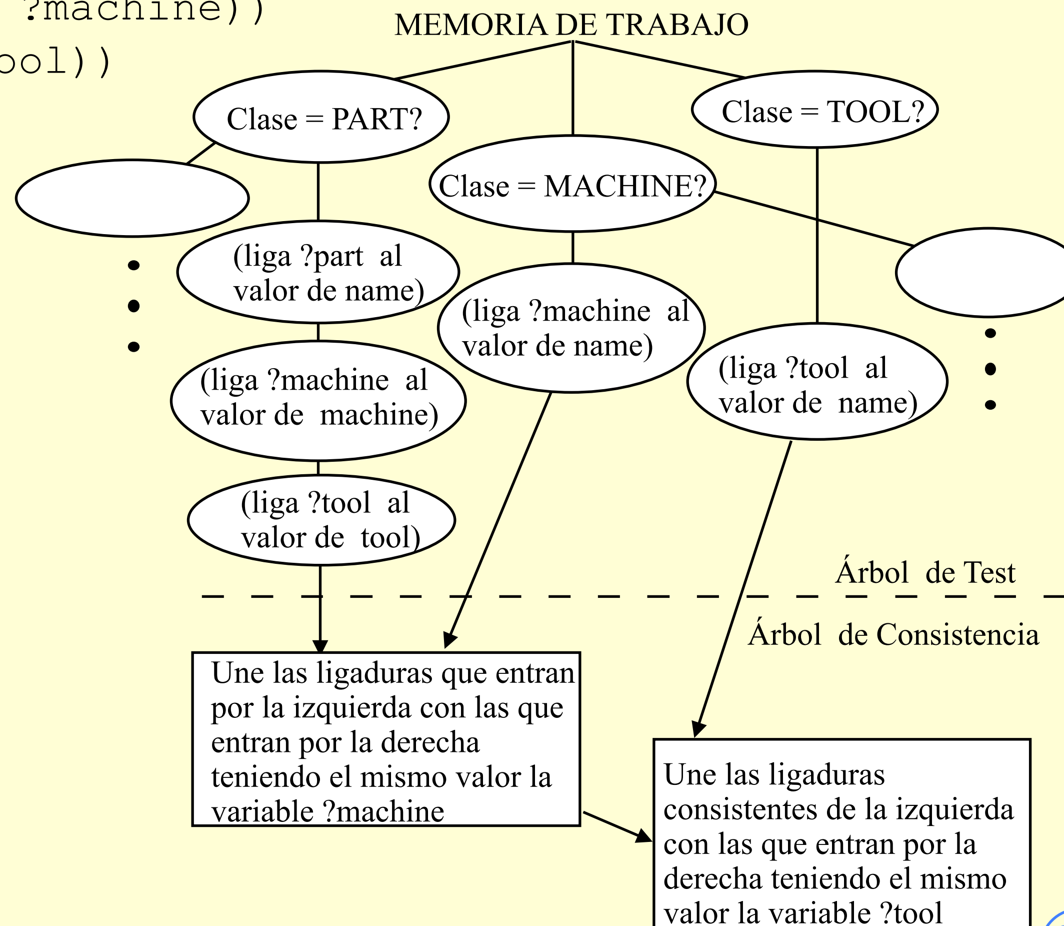



```
CLIPS> (watch compilations)
CLIPS> (load "join.clp")
Defining defrule: compartir-1 +j+j+j+j
Defining defrule: compartir-2 =j=j=j+j
TRUE
```

© José Angel Bañares

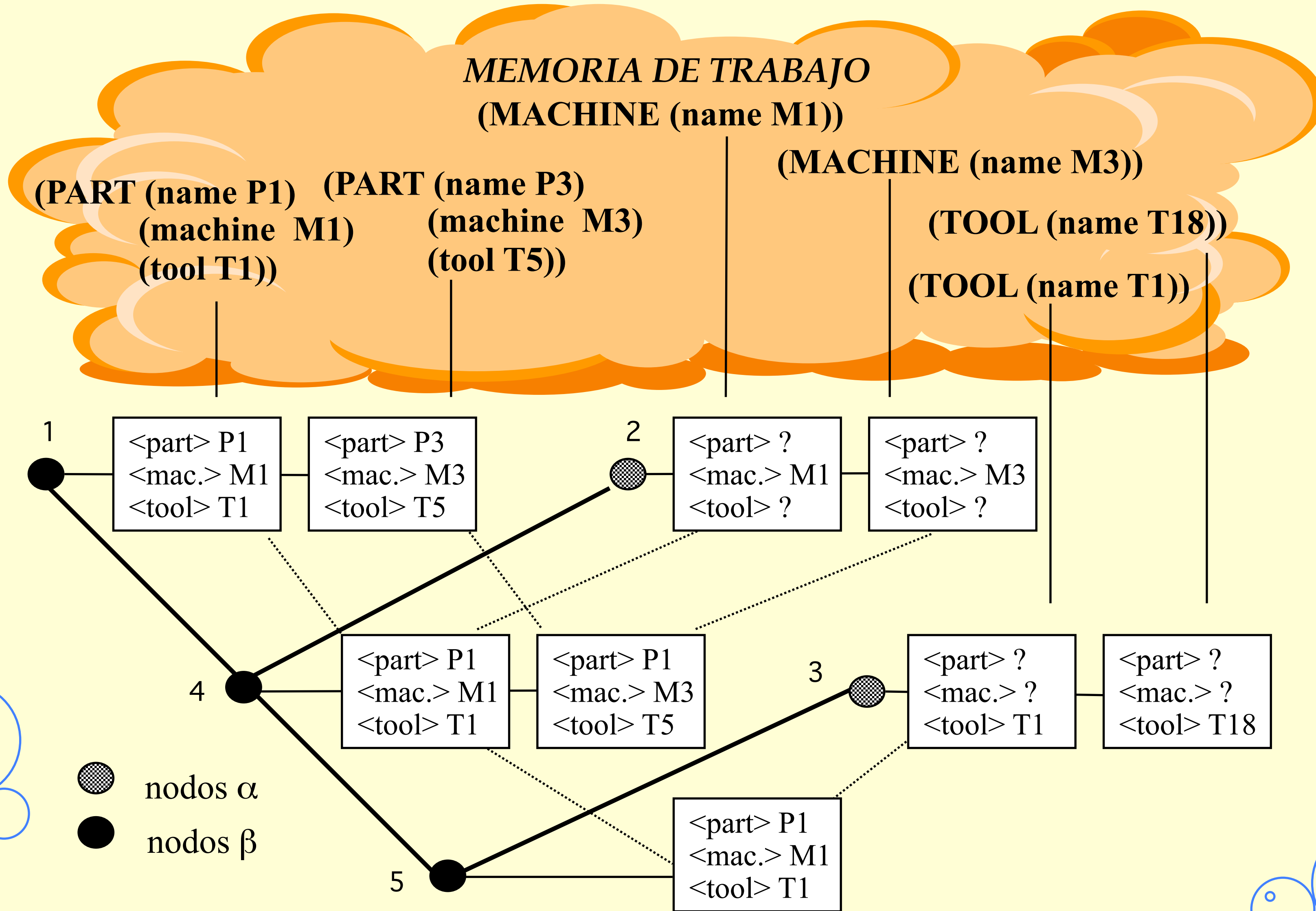
Algoritmo de RETE

```
(defrule load
  (PART (name ?part) (machine ?machine) (tool ?tool))
  (MACHINE (name ?machine))
  (TOOL (name ?tool))
==>
  ...)
```



© José Angel Bañares

α y β memorias



© José Angel Bañares

Algoritmo de TREAT

- **Conjetura**

McDermott, Newell y Moore, 1978

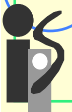
“Recalcular completamente < Mantener la información”

– Críticas al algoritmo de RETE
  - » Alto coste del borrado y modificación
  - » El tamaño de la información almacenada pueden ser elevados (Explosión combinatoria)

- **TREAT**

Miranker 1986

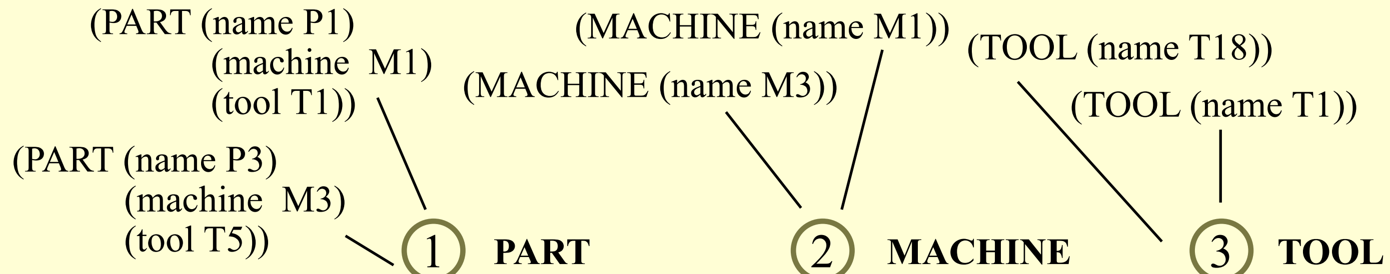
  - » No considera las β-memorias, pero considera el número de condiciones satisfechas.



© José Angel Bañares

Árbol TREAT

Estado Inicial



Partición vieja

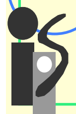
1 PART	2 MACHINE	3 TOOL
<part> P1 <mac.> M1 <tool> T1	<part> ? <mac.> M1 <tool> ?	<part> ? <mac.> ? <tool> T1
<part> P3 <mac.> M3 <tool> T5	<part> ? <mac.> M2 <tool> ?	<part> ? <mac.> ? <tool> T18

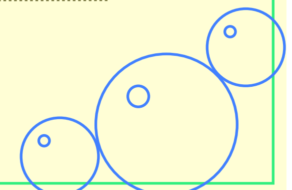
Partición nueva añadir

Partición nueva borrar

Conjunto conflicto

<part> P1 <mac.> M1 <tool> T1





© José Angel Bañares

Añadir en TREAT

+ (PART (name P22) (machine M2) (tool T18))

PART 1 — 2 MACHINE — 3 TOOL

Partición vieja

PART (1)	MACHINE (2)	TOOL (3)
<part> P1 <mac.> M1 <tool> T1	<part> ? <mac.> M1 <tool> ?	<part> ? <mac.> ? <tool> T1
<part> P3 <mac.> M3 <tool> T5	<part> ? <mac.> M2 <tool> ?	<part> ? <mac.> ? <tool> T18

Partición nueva añadir

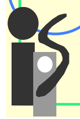
<part> P22
<mac.> M2
<tool> T18

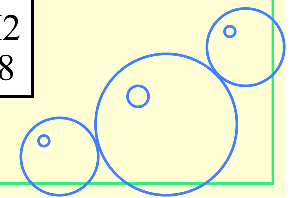
Partición nueva borrar

<part> P22
<mac.> M2
<tool> T18

Conjunto conflicto

<part> P1 <mac.> M1 <tool> T1	<part> P22 <mac.> M2 <tool> T18





© José Angel Bañares

Borrar en TREAT

- (PART (name P22) (machine M2) (tool T18))

PART (1)	(2) **MACHINE**	(3) **TOOL**

Partición vieja

1	2	3
<part> P1 <mac.> M1 <tool> T1	<part> ? <mac.> M1 <tool> ?	<part> ? <mac.> ? <tool> T1
<part> P3 <mac.> M3 <tool> T5	<part> ? <mac.> M2 <tool> ?	<part> ? <mac.> ? <tool> T18
~~<part> P22 <mac.> M2 <tool> T18~~		

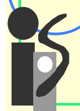
Partición nueva añadir

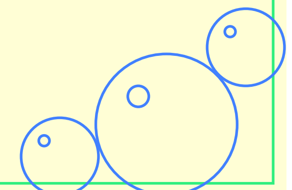
Partición nueva borrar

<part> P22
<mac.> M2
<tool> T18

Conjunto conflicto

~~<part> P22 <mac.> M2 <tool> T18~~	<part> P1 <mac.> M1 <tool> T1

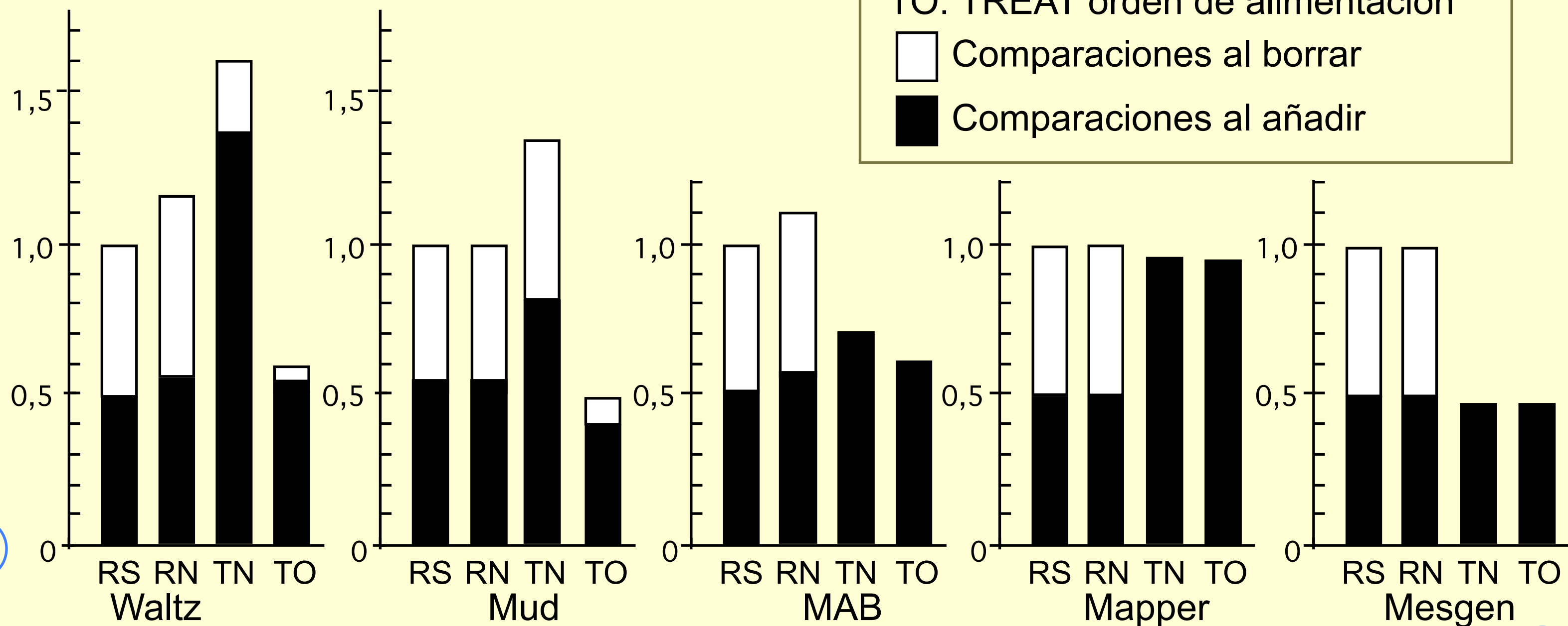


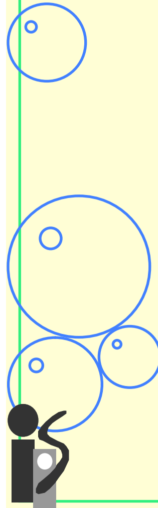


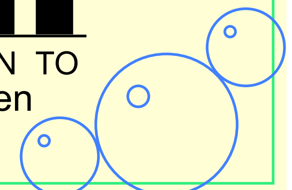
© José Angel Bañares

Comparación Experimental

	Reglas	Condiciones	MT	Ciclos	Agenda
MAB	13	34	11	14	21
Mud	884	2134	241	972	
Waltz	33	130	42	71	193
Mesgen	155	442	34	138	149
Mapper	237	771	1153	84	595







© José Angel Bañares

¿Cómo escribir programas eficientes?

- **Criterios a seguir**
  - Los patrones más específicos primero
  - Colocar al principio patrones que reconocen pocos hechos
  - Patrones que reconocen hecho "volátiles" al final

© José Angel Bañares

La importancia del orden de los patrones

- Evitar una explosión combinatoria de reconocimientos parciales (*partial matches*)

```
(deffacts informacion
  (encuentra a c e g)
  (item a)
  (item b)
  (item c)
  (item d)
  (item e)
  (item f)
  (item g))
```

```
(defrule reconoce-1
   (encuentra ?x ?y ?z ?w)
   (item ?x)
   (item ?y)
   (item ?z)
   (item ?w)
   =>
   (assert (encontrado ?x ?y ?z ?w)))
(defrule reconoce-2
   (item ?x)
   (item ?y)
   (item ?z)
   (item ?w)
   (encuentra ?x ?y ?z ?w)
   =>
   (assert (encontrado ?x ?y ?z ?w)))
```

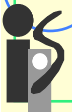
© José Angel Bañares

Ejemplo explosión combinatoria

- **Total: 29** *patrones reconocidos*
  5 *partial matches*

```
CLIPS> (reset)
CLIPS> (matches reconoce-1)
Matches for Pattern 1
f-1
Matches for Pattern 2
f-2 f-3 f-4 f-5 f-6 f-7 f-8
Matches for Pattern 3
f-2 f-3 f-4 f-5 f-6 f-7 f-8
Matches for Pattern 4
f-2 f-3 f-4 f-5 f-6 f-7 f-8
Matches for Pattern 5
f-2 f-3 f-4 f-5 f-6 f-7 f-8
```

```
Partial matches for CEs 1 - 2
f-1,f-2
Partial matches for CEs 1 - 3
f-1,f-2,f-4
Partial matches for CEs 1 - 4
f-1,f-2,f-4,f-6
Partial matches for CEs 1 - 5
f-1,f-2,f-4,f-6,f-8
Activations
f-1,f-2,f-4,f-6,f-8
CLIPS>
```



© José Angel Bañares

Ejemplo explosión combinatoria

```
CLIPS> (reset)
CLIPS> (matches reconoce-2)
Matches for Pattern 1
f-2 f-3 f-4 f-5 f-6 f-7 f-8
Matches for Pattern 2
f-2 f-3 f-4 f-5 f-6 f-7 f-8
Matches for Pattern 3
f-2 f-3 f-4 f-5 f-6 f-7 f-8
Matches for Pattern 4
f-2 f-3 f-4 f-5 f-6 f-7 f-8
Matches for Pattern 5
f-1
Partial matches for CEs 1 - 2
[f-8,f-8],[f-8,f-7],[f-8,f-6]
[f-8,f-5],[f-8,f-4],[f-8,f-3]
[f-8,f-2],[f-2,f-8],[f-3,f-8]
[f-4,f-8],[f-5,f-8],[f-6,f-8]
[f-7,f-8],[f-7,f-7],[f-7,f-6]
[f-7,f-5],[f-7,f-4],[f-7,f-3]
[f-7,f-2],[f-2,f-7],[f-3,f-7]
[f-4,f-7],[f-5,f-7],[f-6,f-7]
[f-6,f-6], ... En total 49
Partial matches for CEs 1 - 3
[f-8,f-8,f-8],[f-8,f-8,f-7],
[f-8,f-8,f-6],[f-8,f-8,f-5],
[f-8,f-8,f-4],[f-8,f-8,f-3],
[f-8,f-8,f-2],[f-8,f-7,f-8],
[f-8,f-7,f-7],[f-8,f-7,f-6]
[f-8,f-7,f-5], ... En total 343
Partial matches for CEs 1 - 4
[f-8,f-8,f-8,f-8],
[f-8,f-8,f-8,f-7],
                ... En total 2401
Partial matches for CEs 1 - 5
f-1,f-2,f-4,f-6,f-8
Activations
f-1,f-2,f-4,f-6,f-8
```

Total

29 ***patrones reconocidos***

2801 ***partial matches***

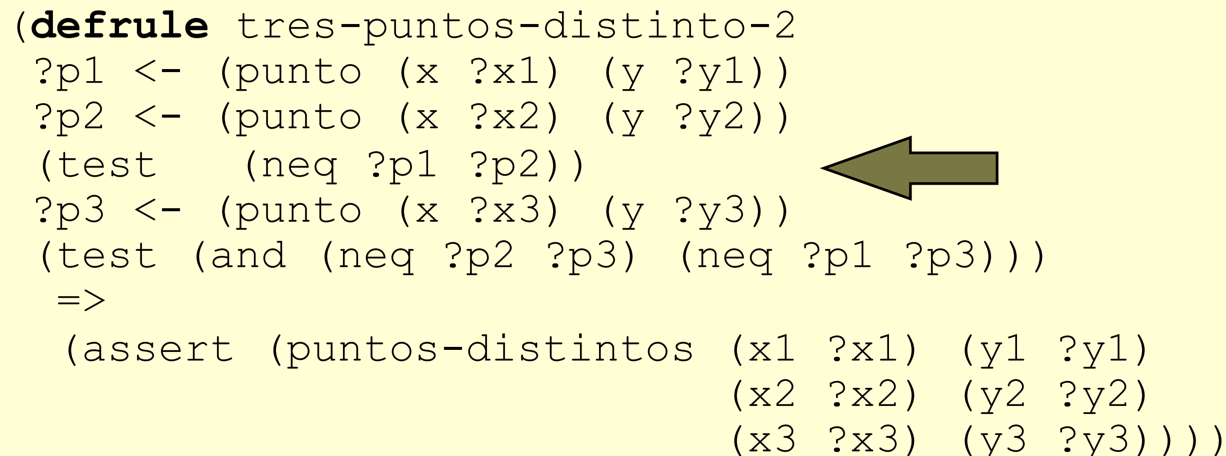
7 + 49 + 343 + 2401 + 1 = 2801

© José Angel Bañares

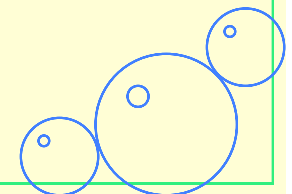
Test en los patrones

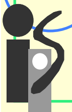
- **Colocar test lo más al principio posible**

```
(defrule tres-puntos-distintos
 ?p1 <- (punto (x ?x1) (y ?y1))
 ?p2 <- (punto (x ?x2) (y ?y2))
 ?p3 <- (punto (x ?x3) (y ?y3))
 (test (and (neq ?p1 ?p2) (neq ?p2 ?p3) (neq ?p1 ?p3)))
  =>
  (assert (puntos-distintos (x1 ?x1) (y1 ?y1)
                            (x2 ?x2) (y2 ?y2)
                            (x3 ?x3) (y3 ?y3))))
```



```
(defrule tres-puntos-distinto-2
 ?p1 <- (punto (x ?x1) (y ?y1))
 ?p2 <- (punto (x ?x2) (y ?y2))
 (test   (neq ?p1 ?p2))
 ?p3 <- (punto (x ?x3) (y ?y3))
 (test (and (neq ?p2 ?p3) (neq ?p1 ?p3)))
  =>
  (assert (puntos-distintos (x1 ?x1) (y1 ?y1)
                            (x2 ?x2) (y2 ?y2)
                            (x3 ?x3) (y3 ?y3))))
```





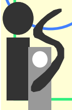
© José Angel Bañares

Restricciones mejor que funciones

- **Más eficiente una restricción sobre un valor que una función**

```
(defrule color-primario
  (color ?x&:(or (eq ?x rojo)
                 (eq ?x verde)
                 (eq ?x azul)))
=>
  (assert (color-pirmario ?x)))

(defrule color-primario
  (color ?x&:rojo|verde|azul)
=>
  (assert (color-primario ?x)))
```



© José Angel Bañares

Variables que ligan varios valores

```
(deftemplate lista (multislot elementos))

(deffacts inicio (lista (elementos a 4 z 2)))

(defrule reconoce-listas
   (lista (elementos $?p $?m $?f))
  =>
   (assert (principio ?p))
   (assert (mitad ?m))
   (assert (final ?f)))
```

Intento	*reconocido por $?p*	*reconocido por $?m*	*reconocido por $?f*
1			a4z2
2		a	4z2
3		a4	z2
4		a4z	2
5		a4z2	
6	a		4z2
7	a	4	z2
8	a	4z	2
9	a	4z2	
10	a4		z2
11	a4	z	2
12	a4	z2	
13	a4z		2
14	a4z	2	
15	a4z2		

© José Angel Bañares

Otras aproximaciones

- **Integración de paradigmas de programación:**
  - El paradigma de programación imperativo

```
(defrule comprueba-si-continua
  ?fase <- (fase comprueba-continuacion)
=>
  (retract ?fase)
  (printout t "Continua (si/no):")
  (bind ?respuesta (read))
  (while (and (neq ?respuesta si) (neq ?respuesta no)) do
        (printout t "Continua (si/no):")
        (bind ?respuesta (read)))
  (if (eq ?respuesta si)
       then (assert (fase continua))
       else (halt)))
```

© José Angel Bañares

Otras aproximaciones

– Representaciones estructuradas del conocimiento:

Fikes y Kehler, 1985

» Integración de Objetos/frames con reglas

- Inferencias automáticas: Herencia, demonios
  - Inferencias embebidas en el esquema de representación
  - Ámbito limitado

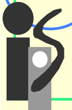
© José Angel Bañares

Otras aproximaciones (frames y reglas)

```
(defmessage-handler persona  agnade-hijos (?hijos)
 (bind ?self:hijos (insert$ ?self:hijos 1 ?hijos)))

(defmessage-handler persona  agnade-hijos after (?
   hijos) ;demonio
   (bind ?conyuge (send ?self get-conyuge))
   (bind ?length (length ?hijos))
   (bind ?i 1)
   (while (<= ?i ?length)
     (bind ?hijo (symbol-to-instance-name (nth$ ?i ?hijos)))
     (send  ?hijo agnade-padres
            (create$ (send ?self get-nombre) ?conyuge))
     (bind ?i (+ 1 ?i))

(defrule hijos-vivos
 (herederos-de ?n)
 ?x <- (object (is-a persona) (vivo no)(nombre ?n))
 (object (is-a persona)(nombre ?h)
         (padres $?P ?nombre $?M)(vivo si))
 =>
 (send ?x agnade-heredero ?h))
```



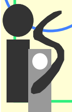
© José Angel Bañares

Otras aproximaciones

- **Filtrado**

  Bronston, Farrell, Kant y Martin 1986

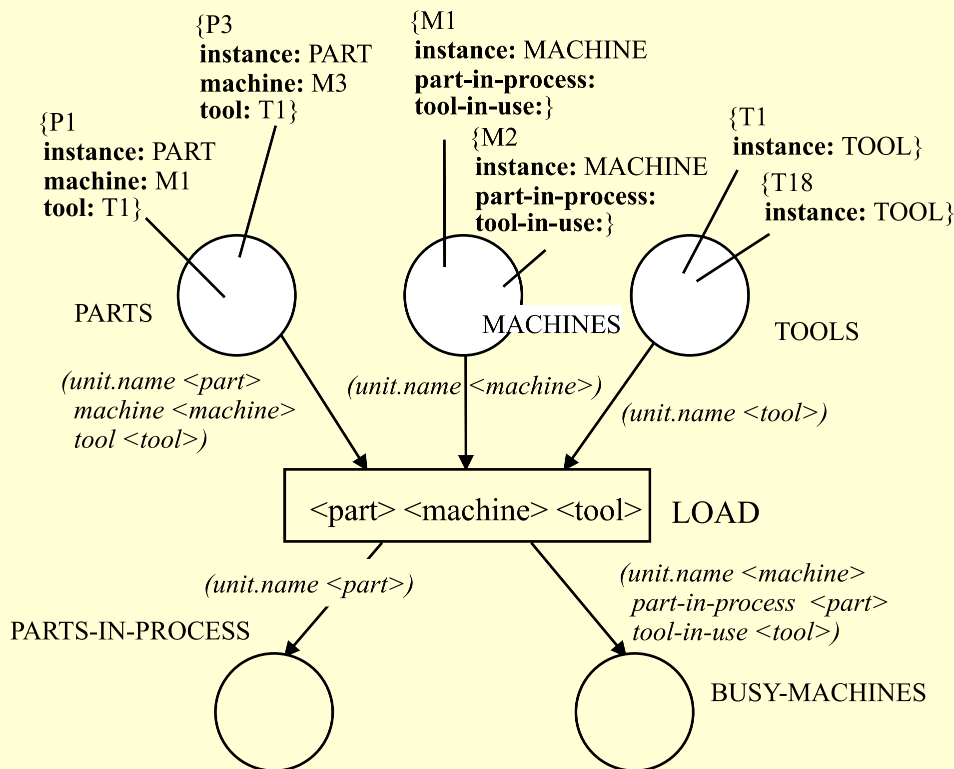
  – Decidir las reglas y los datos que participan en el proceso de reconocimiento:
    - KEE: Las reglas son instancias de clases organizadas en una jerarquía.
    - CLIPS: Reglas organizadas en módulos y visibilidad de hechos
    - OPS5: Filtrado dirigido por objetivos/contextos (MEA)



© José Angel Bañares

Otras aplicaciones: Interpretación de RdP

- **Redes de Petri de alto nivel (RAN)**
  - Las marcas son datos estructurados
  - Similitud entre RAN y sistemas de reglas




```
(PART (name P1)
      (machine M1)
      (tool T1)
      (place PARTS))

(defrule LOAD
  ?p <- (PART (name ?part)
              (machine ?machine)
              (tool ?tool)
  ?m <- (MACHINE (name ?machine)
                 (place MACHINES))
  ?t <- (TOOL (name ?tool)
              (place TOOLS))
 =>
  (modify ?p
          (place PARTS-IN-PROCESS))
  (modify ?m
          (part-in-process ?part)
          (tool-in-use ?tool)
          (place BUSY-MACHINES))
  (modify ?t (place NIL))
```

© José Angel Bañares

Interpretación de RAN

{P3
instance: PART
machine: M3
tool: T1}

{M1
instance: MACHINE
part-in-process:
tool-in-use:}

{P1
instance: PART
machine: M1
tool: T1}

{M2
instance: MACHINE
part-in-process:
tool-in-use:}

{T1
instance: TOOL}

{T18
instance: TOOL}

PARTS

MACHINES

TOOLS

(unit.name <part>
machine <machine>
tool <tool>)

(unit.name <machine>)

(unit.name <tool>)

<part> <machine> <tool>

LOAD

(unit.name <part>)

(unit.name <machine>
part-in-process <part>
tool-in-use <tool>)

PARTS-IN-PROCESS

BUSY-MACHINES

{P3
instance: PART
machine: M3
tool: T1}

{M2
instance: MACHINE
part-in-process:
tool-in-use:}

{T18
instance: TOOL}

(unit.name <part>
machine <machine>
tool <tool>)

(unit.name <machine>)

(unit.name <tool>)

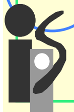
<part> <machine> <tool>

(unit.name <part>)

(unit.name <machine>
part-in-process <part>
tool-in-use <tool>)

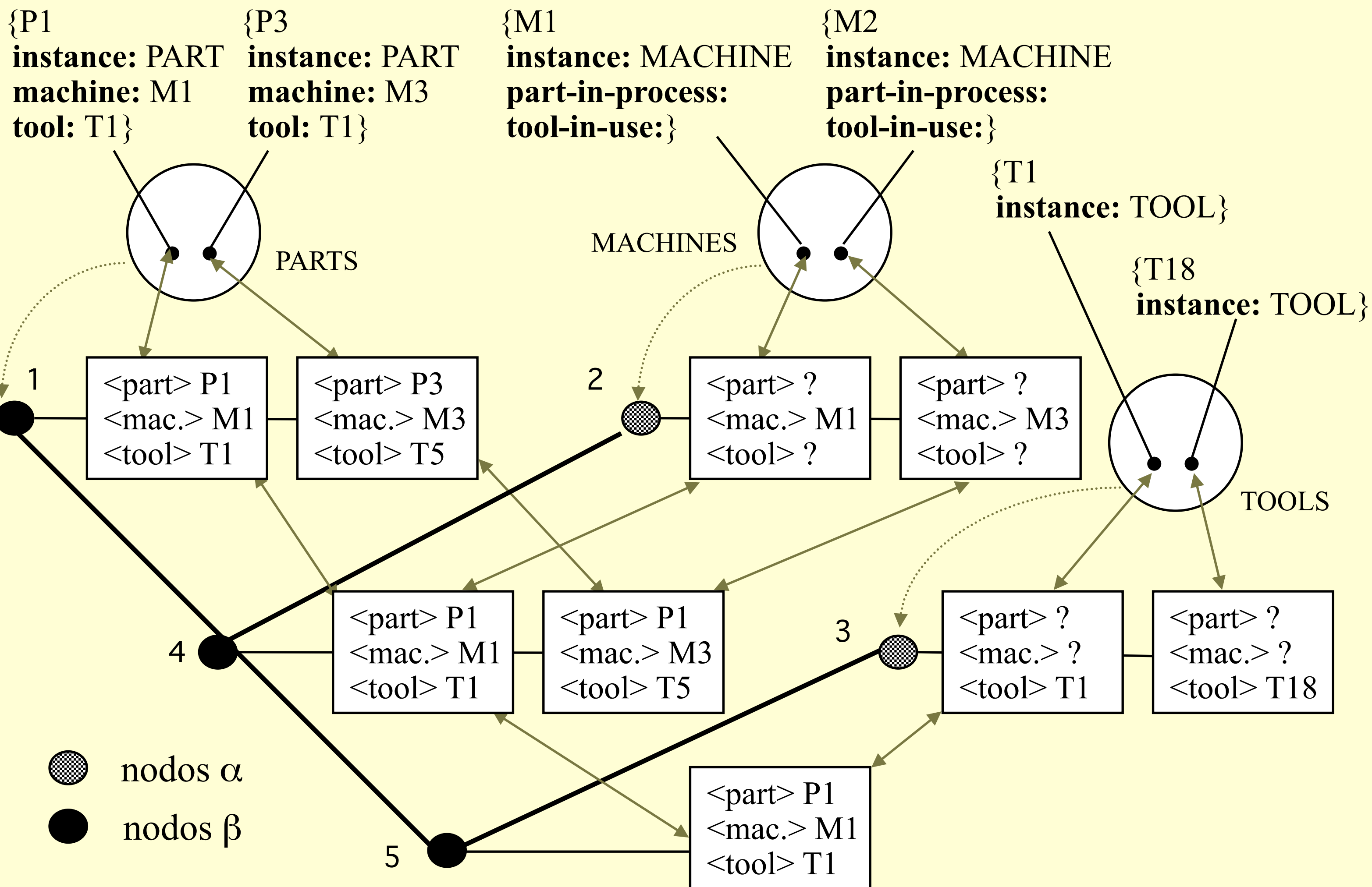
{P1
instance: PART
machine: M1
tool: T1}

{M1
instance: MACHINE
part-in-process: P1
tool-in-use: T1}



© José Angel Bañares

Árboles RETE



© José Angel Bañares

Árboles TREAT

{P1
instance: PART
machine: M1
tool: T1}

{P3
instance: PART
machine: M3
tool: T1}

{M1
instance: MACHINE
part-in-process:
tool-in-use:}

{M2
instance: MACHINE
part-in-process:
tool-in-use:}

{T1
instance: TOOL}

{T18
instance: TOOL}

PARTS

MACHINES

TOOLS

1

<part> P1
<mac.> M1
<tool> T1

<part> P3
<mac.> M3
<tool> T5

2

<part> ?
<mac.> M1
<tool> ?

<part> ?
<mac.> M3
<tool> ?

3

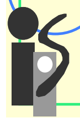
<part> ?
<mac.> ?
<tool> T1

<part> ?
<mac.> ?
<tool> T18

<part> P1
<mac.> M1
<tool> T1

nodos α

Modos de disparo



© José Angel Bañares

Comparación experimental

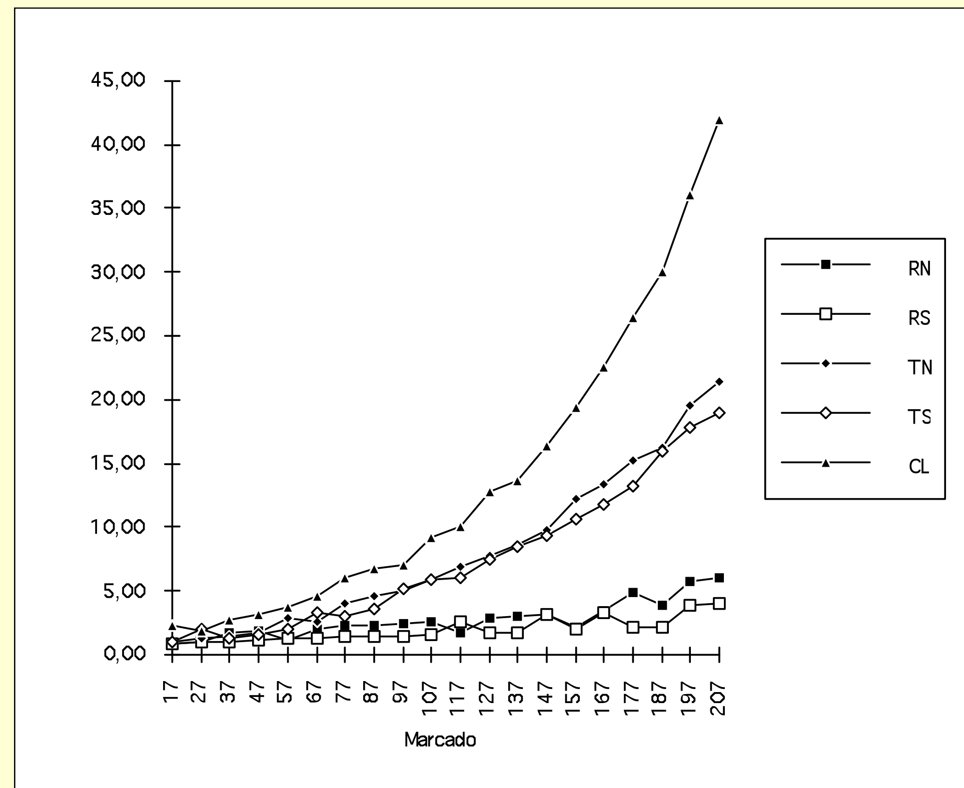
RN: RETE, evitando realizar procesos de búsqueda para eliminar patrones

RS: RETE, limitando las búsquedas a las β-memorias

TN: TREAT, sin ningún proceso de búsqueda

TS: TREAT, con búsqueda de los modos de disparo a eliminar

CL: Una implementación clásica en RdpN (Condition membership)



© José Angel Bañares

Referencias

- C. L. Forgy
  Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem
  Artificial Intelligence, N . 19, 1982, 17-37
- D. P. Miranker
  TREAT: A Better Match Algorithm for AI Production Systems
  Proc. of AAAI 87 Conference on AI, 1987, 42-47
- L. Brownston, R. Farrell, E. Kant, N. Martin
  Programming Expert Systems in OPS5. An Introduction to Rule-Based Programming
  Addison-Wesley, 1985, Capítulo 6, Sección 7.3.1
- J. Giarratano, G. Riley
  Expert Systems, Principles and Programming. Third Edition.
  International Thomson Company , 1998, Capítulo 11