



***Manejo del conocimiento...***

En un “sistema real” es necesario poner atención a muchos detalles, por ejemplo:

- ◆ Agregar a la respuesta el conteo de tuplas...
   
 $(:\text{count} (\exists (\text{atr}_1 . \text{val}_1) . . . (\text{atr}_m . \text{val}_m))$
- ◆ Consultas sobre la presencia de un atributo en un objeto...
   
 $(\exists (\text{attribute} . \text{value}))$

Si una consulta existencial responde falso, no indica nada sobre la presencia del atributo consultado...

$(:\text{feature-present} \text{ attribute})$

*Dr. Salvador Godoy Calderón*

***Manejo del conocimiento...***

En un “sistema real” es necesario poner atención a muchos detalles, por ejemplo:

- ◆ Consultas sobre valores sin necesidad de especificar la etiqueta del atributo...
   
 $(\exists ?var = \text{azul})$ 
  
 $(\exists \text{azul} \text{ pequeño volador})$
- ◆ Generación de expresiones de consulta a alguna base de datos...
- ◆ Etc...

*Dr. Salvador Godoy Calderón*

### Cuantificador universal..

Las variables lógicas del lenguaje usado representan a los objetos en el universo de estudio... en este caso los objetos son las tuplas de conocimiento...

Por tanto, una expresión cuantificada universalmente,

$$(\forall x) [P_1(x) \wedge P_2(x) \wedge \cdots \wedge P_n(x)]$$

Se refiere a TODAS las tuplas de conocimiento, lo cual puede ser muy difícil que ocurra...

Dr. Salvador Godoy Calderón

### Simplificación...

Una alternativa es reducir el ámbito de cuantificación...

En lugar de buscar una condición que se cumpla en TODAS las tuplas de conocimiento, buscamos que se cumpla en un subconjunto bien definido de ellas...

$$(\forall x \in S) [P_1(x) \wedge P_2(x) \wedge \cdots \wedge P_n(x)]$$

$$(\forall x) P(x) \rightarrow [Q_1(x) \wedge Q_2(x) \wedge \cdots \wedge Q_n(x)]$$

$$p \rightarrow q$$

$$(\rightarrow p \ q)$$

*(:for-all ((attr1 . val1) (attr1 . val1)...)) ((attr1 . val1) (attr1 . val1)...))*

Dr. Salvador Godoy Calderón

*Forma correcta...*

Si se simplifican todas las consultas con cuantificador universal, entonces la sintaxis de las consultas es:

$$(\exists x) P(x)$$

(:*exists* (attr1 . val1) (attr2 . val2) ... )

$$(\nexists x) P(x)$$

(:*not-exists* (attr1 . val1) (attr2 . val2) ... )

$$(\forall x) P(x)$$

$$(\forall x) P(x) \rightarrow Q(x)$$

(:*for-all* ((attr1 . val1) (attr2 . val2) ...) ((attr1 . val1) (attr2 . val2) ...) )

$$(\forall x) P(x)$$

$$(\forall x) P(x) \rightarrow Q(x)$$

(:*not-for-all* ((attr1 . val1) (attr2 . val2) ...) ((attr1 . val1) (attr2 . val2) ...) )

Dr. Salvador Godoy Calderón

**La sesión  
anterior...**

## Lenguaje lógico...

```
(gusta mauricio música)
(amigos bill (padre roberto))
(color bloque2I rojo)
```

```
(rule :name color-de-objetos-en-caja2
  :if  (dentro-de bloque1 caja2)
       (estado caja2 abierta)
  :then (color bloque1 verde))
```

Las variables sólo pueden ser usadas como argumento de predicados y funciones; no pueden ser etiquetas de predicados ni funciones, conectores lógicos, ni argumentos de los conectores...

Dr. Salvador Godoy Calderón

## Planteamiento...

Las expresiones que incluyen variables están implícitamente cuantificadas por el cuantificador universal.

Para cuantificar existencialmente hay dos soluciones:

- ♦ *Skolemización*: substituir la variable a cuantificar existencialmente por una constante y replantear el conocimiento
- ♦ *Expresiones funcionales*: substituir la variable a cuantificar existencialmente por una expresión funcional con argumentos constantes y replantear el conocimiento

Dr. Salvador Godoy Calderón

## *Especificación...*

Usaremos sólo dos funciones para construir un intérprete de reglas:

**(store <expresión>)**

Para añadir (sin validar, por el momento) **<expresión>** a la memoria declarativa...

**(retrieve <expresión>)**

Para realizar la consulta representada por **<expresión>** ...

En general, las expresiones de consulta pueden contener variables, mientras que el conocimiento declarativo no...

*Dr. Salvador Godoy Calderón*

## *Encadenamiento...*

Si existe un hecho que unifique directamente con la consulta, la respuesta es inmediata...

De lo contrario:

- ◆ Buscar todos los **consecuentes** de regla que unifiquen con la consulta...
- ◆ Substituir el resultado de la unificación en los **antecedentes** de cada regla...
- ◆ La expresión resultante es una nueva consulta...
- ◆ Continuar recursivamente hasta lograr respuesta inmediata...

*Dr. Salvador Godoy Calderón*



### Generalidades...

Un “recuperador deductivo” es una versión reducida de un “motor de inferencia” y generalmente incluye capacidades para:

- ♦ Editar hechos y reglas...
- ♦ Realizar consultas a la BC...
- ♦ Unificar expresiones...
- ♦ Realizar encadenamiento hacia atrás...
- ♦ Realizar encadenamiento hacia adelante...

Usaremos el motor de inferencia de *Chris Riesbeck* de la Universidad de Northwestern, USA.



**Chris Riesbeck, Associate Professor**

Room 3315  
Ford Motor Company Engineering Design Center  
2133 Sheridan Road, Evanston, IL 60208

email: c-riesbeck@northwestern.edu Phone: 847-491-7279

Dr. Salvador Godoy Calderón

### *La Mecánica...*

El profesor Riesbeck tiene dos versiones de su recuperador deductivo:

- ◆ La versión simplificada (**retriever.lisp**) sólo incluye encadenamiento inverso de las reglas...
- ◆ La versión extendida (**ddr.lisp**) incluye también aserciones simples y encadenamiento directo...

En estos programas la función **store** se llama **tell** y la función **retrieve** se llama **ask**...

*Dr. Salvador Godoy Calderón*

### *Capacidades...*

El recuperador deductivo maneja un lenguaje de representación que permite expresar:

- ◆ Hechos (aserciones)  
**(vive-en tigre selva-tropical)**
- ◆ Reglas para encadenar hacia adelante:  
**(-> (depredador-de ?x ?y) (presa-de ?y ?x))**
- ◆ Reglas para encadenar hacia atrás:  
**(<- (soporta-humedad ?x) (vive-en ?x selva) )**

*Dr. Salvador Godoy Calderón*

## *La mecánica...*

Las funciones principales del motor de inferencia son:

- ♦ **(tell <expresión>)** Agrega **<expresión>** a la base de conocimiento (declarativo y procedural)...
- ♦ **(ask <expresión>)** Consulta la BC incluyendo inferencia...
- ♦ **(clear-rule-base)** Reinicia vacía la BC...

Y una sintáxis que separa las reglas para ser encadenadas hacia atrás y aquellas para encadena hacia adelante...

$$(<- q p_1 p_2 p_3 \dots) \quad (-> p_1 p_2 \dots) \quad (-> (and p_1 p_2 \dots) q)$$

Dr. Salvador Godoy Calderón

## *La mecánica...*

La función **tell** se usa para agregar, tanto hechos como reglas (para encadenamiento hacia adelante), a la BC...

- ♦ Si se agrega una regla, la regla se almacena...
- ♦ Si el hecho unifica con alguna regla de tipo  
 $(-> p_1 p_2 \dots)$   
entonces se agregan a la BC todos los **q1 q2...**
- ♦ Si el hecho unifica con **p1** en alguna regla de tipo  
 $(-> (and p_1 p_2 \dots) q_1 q_2 \dots)$   
y los demás términos del antecedente se pueden encontrar con **ask**, entonces se agregan a la BC todos los **q1 q2...**

Dr. Salvador Godoy Calderón

### Algunos ejemplos...

```
(tell '(-> (esposo ?x ?y) (esposa ?y ?x)))
(tell '(esposo Jorge Mariana))
```

En el primer caso, se agrega a la BC sólo la regla (etiquetada para encadenamiento hacia adelante)...

En el segundo caso se agregan a la BC:

```
(esposo Jorge Mariana)
(esposa Mariana Jorge)
```

*Dr. Salvador Godoy Calderón*

### La mecánica...

La función **ask** se usa para consultar la BC, incluyendo inferencias con reglas (para encadenamiento hacia atrás)...

```
(tell '<- (es-caballo ?y) (padre-de ?x ?y) (es-caballo ?x)) )
(tell '(es-caballo Furia))
(tell '(padre-de Furia Rosinante))
```

```
(ask '(es-caballo Furia))
(ask '(es-caballo Rosinante))
(ask '(es-caballo ?x))
```

*Dr. Salvador Godoy Calderón*

### *Un detalle importante...*

Se proporcionan dos funciones para rastreo:

(**ask-trace <consulta> [<expresión>]**)  
(**show-trace**)

**ask-trace** crea un árbol del proceso de inferencia involucrado en la consulta...

**show-trace** muestra el árbol previamente creado...

consultar

<https://www.cs.northwestern.edu/academics/courses/325/readings/ddr.php>

Dr. Salvador Godoy Calderón

### **La tarea...**



**TAREA...**

Usando el recuperador deductivo de *Riesbeck*, plantear los hechos y reglas necesarios para resolver el problema de la zebra:

Life International magazine, Diciembre 17, 1962.  
Soluciones en el número de Marzo 25, 1963.

Conjunto residencial, 5 casas, cada propietario con diferente nacionalidad, mascota, bebida y cigarros...

- |   |  |
|---|--|
| 1) Existen 5 casas diferentes<br>2) El dueño inglés vive en la casa roja<br>3) El español tiene un perro<br>4) En la casa verde se bebe café<br>5) El ucraniano bebe té.<br>6) La casa verde está junto a la blanca<br>7) El fumador de Old Gold tiene un caracol<br>8) En la casa amarilla se fuma Kools<br>9) En la casa a la mitad se bebe leche<br>10) El noruego vive en la primera casa | 11) El fumador de Chesterfields vive junto al dueño que tiene un zorro<br>12) Los cigarros Kools se fuman en la casa junto a la que tiene un caballo<br>13) El fumador de Lucky Strike bebe jugo<br>14) El dueño japonés fuma Parliaments.<br>15) El noruego vive junto a la casa azul |
|---|--|

¿Quién bebe agua?  
¿Quién tiene una zebra?

*Dr. Salvador Godoy Calderón*

**Condiciones...**

- ♦ Estrictamente basta con dar una solución a la tarea usando el recuperador simplificado...
- ♦ Quien solucione la tarea con el recuperador extendido y encadenando reglas hacia adelante tendrá 3 puntos extra en la tarea.
- ♦ Tanto las condiciones iniciales del problema, como el conjunto de reglas para la solución se deben cargar de archivo. Esto requiere algunas modificaciones al archivo con el código fuente...

*Dr. Salvador Godoy Calderón*

