Universidad de Los Andes

http://cesimo.ing.ula.ve

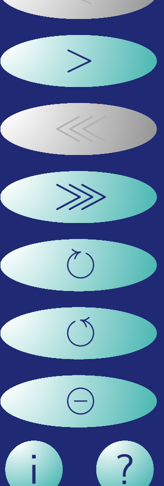
Unidad 3: Semánticas para programas lógicos definidos

Jacinto Dávila

mailto:jacinto@ula.ve

Centro de Simulación y Modelos (CESIMO)

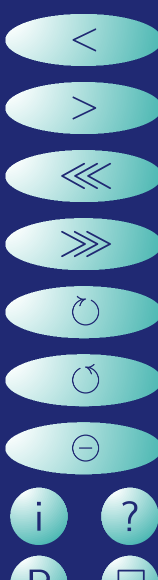
ULA LÓGICA



Esta unidad es acerca de una *teoría de prueba* y una teoría de modelos para una lógica computacional. En particular discutiremos:

1. Resolución SLD.
2. Conceptos básicos en las semánticas para programas lógicos definidos.
   a) modelos mínimos.
   b) la matemática del punto fijo y la función de significados.
   c) la coincidencia de las semánticas.

ULA LÓGICA



¿Recuerdan la Regla de Resolución?

Este es un diagrama (árbol) ilustrando como usar esa regla en una prueba formal:

$C_1 : \neg\ p(x)$

$C_2 : p(y) \vee \neg q(y)$

$C_3 : q(a)$

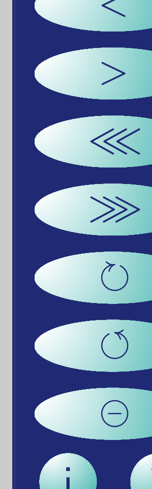
C_1..............C_2..C_2.............C_3

....$R_1 : \neg\ q(y)$ ....C_3......................................$R_3 : p(a)$.........C_1

............R_2: $\Box$..$R_4 : \Box$

Nota: los puntos no significan nada en este lámina.

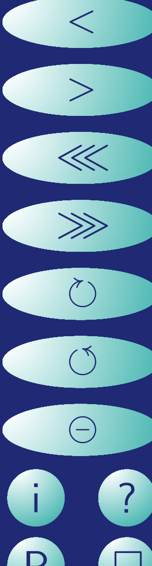
ULA LÓGICA



Un ejercicio para ayudar la memoria

Actividad: En ciertos países todos ayudan a aquellos que no pueden ayudarse ellos mismos. Pruebe que esos son países de la AUTO PROTECCIÓN (AUTO -AYUDA). (Pista: Use resolución para mostrar que cada persona se ayuda a sí misma.) [Hogger:1990]

ULA LÓGICA

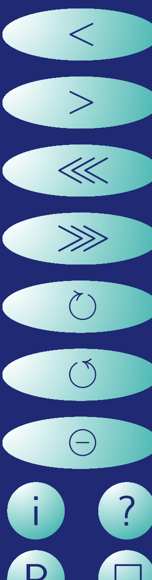


Resolución SLD

Como dijimos en su oportunidad, *Resolución* es sólo una regla de inferencia. Si queremos un algoritmo para prueba automática de teorema y *problem solving*, tendremos que acompañar la regla con ciertas estrategias para decidir cuál cláusula y cuál literal se usará en cada paso de resolución.

Una de esas combinaciones de Resolución con estrategias computacionales se llama Resolución SLD y es una forma restringida de la regla general que se aplica sólo para resolver cláusulas negativas (preguntas) con programas lógicos de cláusulas definidas únicamente.

ULA LÓGICA



El acrónimo SLD

La **S** indica que se está usando una regla fija para seleccionar el literal sobre el que operará el paso de resolución (una regla de computación).

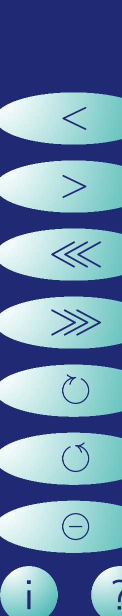
La **L** indica que el proceso es lineal puesto que siempre se usa como padre en el paso de resolución al más reciente resolvente (esta es la cláusula central, la cláusula lateral es precisamente la que se toma del programa usando la regla de computación).

La **D** indica que los programas son conjuntos de cláusulas definidas.

La nueva regla restringida (Resolución SLD) sigue siendo (como Resolución) completa con respecto a refutación (¿ Qué significa esto?).

Observe que cuando se habló de una regla fija para seleccionar el literal no se dijo cuál regla. Resolución sigue siendo completa con la estrategia SLD sin importar cual regla de computación se usa (a esto se le llama independencia de la regla de computación).

ULA LÓGICA



Definiciones formales en resolución SLD

Definición lm3-1: Una *substitución* es un conjunto finito con la forma

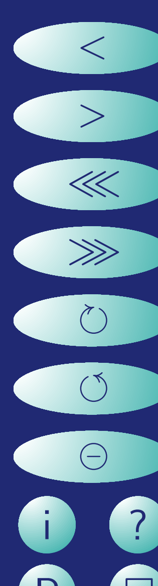
$$\{ V_1/t_1 ,...., V_\mathrm{n}/t_\mathrm{n} \},$$

donde cada V_i es una variable distinta y cada término t_i es distinto de su V_i.

Definición lm3-2 : Si t es un término y θ es una substitución $t\theta$ es el resultado de reemplazar cada V_i en t por un t_i tal que $V_i/t_i \in \theta$. De la misma manera, si $\Phi\theta$ (donde Φ es una fórmula) es el resultado de reemplazar cada V_i en Φ por un t_i tal que $V_i/t_i \in \theta$.

Definición lm3-3 : La substitución θ es un unificador de dos átomos Φ_1 y Φ_2 si $\Phi_1\theta$ es idéntica a $\Phi_2\theta$. Si tal unificador existe, entonces Φ_1 y Φ_2 se consideran *unificables.*

ULA LÓGICA



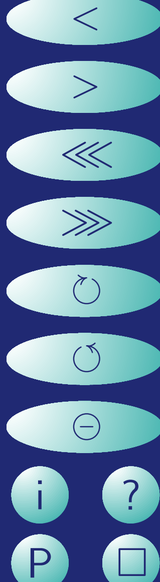
Definición lm3-4: Un unificador θ_1 de 2 átomos Φ_1 y Φ_2 es el *unificador más general* para esos 2 átomos, si para cada unificador θ_2 de Φ_1 y Φ_2 existe una substitución θ_3 tal que

$$\Phi_1\theta_1\theta_3 = \Phi_1\theta_2.$$

Definición lm3-5: Una cláusula Φ_1 es una *variante* de una cláusula Φ_2 si existen substituciones θ_1 y θ_2 tal que $\Phi_1\theta_1 = \Phi_2$ y $\Phi_1 = \Phi_2\theta_2$.

Definición lm3-6: La cláusula objetivo : $\leftarrow \Phi_1\theta \wedge \ldots\ldots \wedge \Phi_{k-1}\theta \wedge \Psi_2\theta \wedge \Psi_m\theta \wedge \Phi_{k-1}\theta \wedge \ldots\ldots \wedge \Phi_n\theta$. es una *resolvente* de la cláusula objetivo: $\leftarrow \Phi_1 \wedge \Phi_2 \wedge \ldots\ldots \wedge \Phi_n$, con la cláusula definida $\Psi_1 \leftarrow \Psi_2 \wedge \ldots \Psi_m$ usando el u.m.g. θ de Φ_k y Ψ_1.

ULA LÓGICA



Refutación SLD con un ejemplo

Definición lm3-7: Sea T la conjunción de un conjunto de cláusulas definidas y sea O_o una cláusula objetivo. Una **refutacion-SLD** de O_o usando T es una secuencia de cláusulas objetivo O_o a O_n y secuencia de substitución θ_1 a θ_n donde:

- O_n es la cláusula vacia y
- cada O_{i+1} es un resolvente de O_i con una variante de una cláusula en T usando subs. θ_{i+1}.

Considere este ejemplo sobre el siguiente programa T.

Sea T :

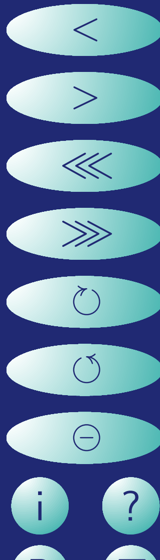
$infeliz(X) \leftarrow mortal(X) \wedge iniluminado(X).$

$mortal(X) \leftarrow humano(X).$

$humano(socrates).$

$iniluminado(socrates).$

ULA LÓGICA



La siguiente es una refutación SLD del objetivo ← infeliz (socrates) usando T:

$\leftarrow infeliz(socrates)$

$\downarrow$

$\leftarrow mortal(socrates) \wedge iniluminado(socrates)$

$\downarrow$

$\leftarrow humano(socrates) \wedge iniluminado(socrates)$

$\downarrow$

$\leftarrow iniluminado(socrates)$

$\downarrow$

$\square$

Actividad: Compare esta refutacion con la traza PROLOG del mismo programa y pregunta.

Una refutación ES un argumento.

ULA LÓGICA

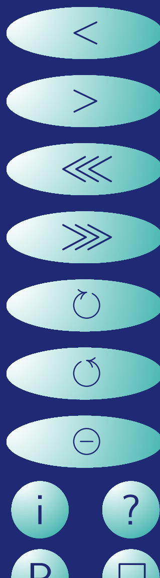
Resolución SLD y el significado de “computar”.

Definición lm3-8: Si θ_1= { $V_1/t_1, ...,\ V_n/t_n$} y θ_2 = { $W_1/K_1,...,\ W_m/K_m$} son substituciones, entonces la composición $\theta_1\theta_2$ de θ_1 y θ_2 es la substitución $\theta_3 - \theta_4$ donde θ_3 = { $V_1/t_1\ \theta_2$,...., $V_n/t_n\ \theta_2$ } $\cup\ \theta_2$ y θ_4 es el conjunto de todos los V_i/t_i tal que $V_i = t_i\ \theta_2$ y todos los W_j/K_j tal que $W_j \in$ { $V_1....V_n$ }

Definición lm3-9: Sea T la conjunción de un conjunto de cláusulas definidas y sea O una cláusula objetivo . El conjunto de todos los V/t tal que V aparece en O y V/t $\in \theta_1, ..., \theta_n$ (donde $\theta 1$ a θ_n es la secuencia de substituciones en una refutación-SLD de O usando T), es una *substitución respuesta* de O a partir de T.

Computar ES aplicar la regla de inferencia

ULA LÓGICA



$$\leftarrow infeliz(X)$$

$$\downarrow$$

$$\leftarrow mortal(X) \wedge iniluminado(X)$$

$$\downarrow$$

$$\leftarrow humano(X) \wedge iniluminado(X)$$

$$\downarrow \theta = \{X/socrates\}$$

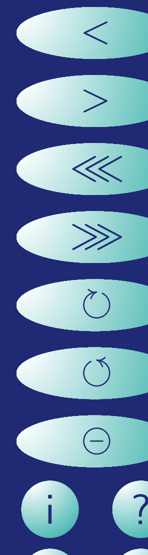
$$\leftarrow iniluminado(socrates)$$

$$\downarrow$$

$$\square$$

En este caso el valor de X es una respuesta computada

ULA LÓGICA



Los resultados claves acerca de resolución SLD

Teorema lm3-1 (resolución SLD es sana) : Si existe una refutación SLD de $?\Phi_1 \wedge \Phi_2 \wedge ..... \wedge \Phi_n$ a partir de T usando substitución θ, entonces:

$$\mathrm{T} \models \Phi_1\theta \wedge \Phi_2\theta \wedge ..... \wedge \Phi_n\theta$$

Teorema lm3-2 (resolución SLD es completa) : Si

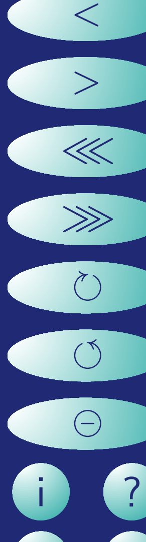
$$\mathrm{T} \models \Phi_1\theta_1 \wedge \Phi_2\theta_1 \wedge ..... \wedge \Phi_n\theta_1$$

donde θ_1 es una substitución, entonces existe una substitución θ_2 y una refutación SLD de $?\Phi_1 \wedge \Phi_2 \wedge ..... \wedge \Phi_n$ a partir de T con substitución respuesta θ_3 tal que

$$\Phi_i\theta_1 = \Phi_i\theta_3\theta_2 \text{ para cada } i$$

Pruebas en [Lloyd, 1987: Foundations of Logic Programming].

ULA LÓGICA



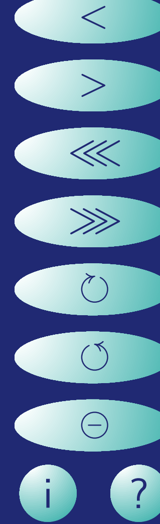
Un sistema de programación lógica

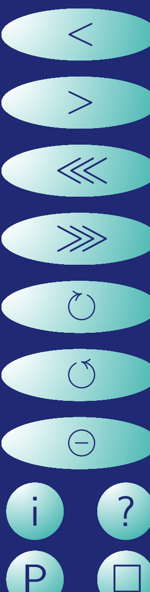
Observen que resolución (incluso la SLD) no prescribe una forma particular de responder estas preguntas :

- ¿Qué átomo en una cláusula objetivo debemos usar en la resolución?
- ¿Cuál cláusula definida se debe usar para la resolución?

Definición lm3-10: Una *función de selección* (ó estrategia de selección) es una función que señala, en una conjunción de átomos, a uno de esos átomos (el *átomo seleccionado*).

ULA LÓGICA





Elementos de un sistema de programación lógica

Teorema lm3-3: Para cualquier función de selección, si es el caso que

$$\mathrm{T} \models \Phi_1\theta_1 \wedge ....\Phi_n\theta_1 \text{ donde } \theta_1 \text{ es una sustitución.}$$

entonces existe una sustitución θ_2, una refutación-SLD de O usando T y sustitución respuesta θ_3 tal que :

$$\Phi_i\theta_1 = \Phi_i\theta_3\theta_2$$

Este teorema dice que resolución SLD computa la respuesta más general (menos comprometida) posible.

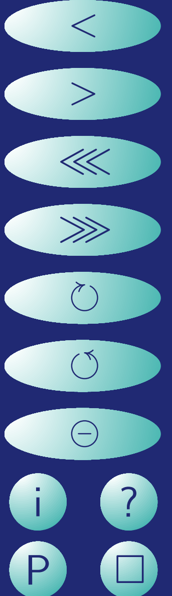
ULA LÓGICA

PROLOG es un sistema de programación lógica. No es el único (busque otros en Internet). Tampoco puede decirse que es el mejor. Es el más maduro y ha servido para una enorme cantidad de ejercicios de investigación y aplicaciones comerciales.

PROLOG extiende SLD con las siguientes estrategias :

- Siempre selecciona el átomo *más a la izquierda* para la siguiente resolución.
- Las cláusulas definidas son "usadas" en el orden en el que aparecen en el texto (de arriba hacia abajo) (o de izq. a derecha) y se emplea una **estrategia de búsqueda** PRIMERO EN PROFUNDIDAD.

ULA LÓGICA

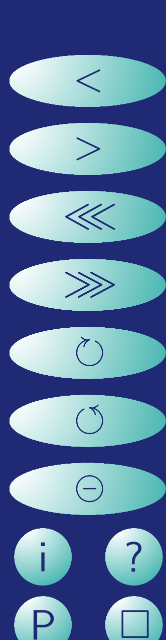


Otro ejemplo de cómo trabaja Resolución SLD

$infeliz(X) \leftarrow mortal(X) \wedge iniluminado(X)$	C1
$mortal(X) \leftarrow humano(X)$	C2
$humano(buda)$	C3
$humano(socrates)$	C4
$iniluminado(socrates)$	C5

Este es un caso de **Backtracking** (*salto atrás*) al fallar la prueba en una rama.

ULA LÓGICA



$\leftarrow$ *infeliz (X)*

$\downarrow$

$\leftarrow$ *mortal (X)* $\wedge$ *iniluminado (X)*

$\downarrow$

$\leftarrow$ *humano (X)* $\wedge$ *iniluminado(X)*

$\downarrow$ X=buda $\searrow$X=socrates.

$\downarrow$

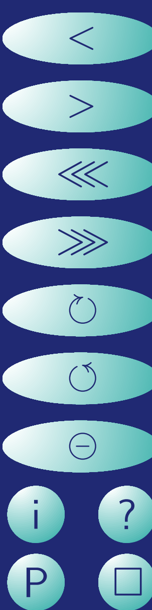
$\leftarrow$ *iniluminado(buda)* $\leftarrow$*iniluminado(socrates)*

$\downarrow$ $\downarrow$

$\bullet$ $\square$

Actividad: Compare este árbol con la traza PROLOG sobre el mismo problema y explique cómo recorre PROLOG el árbol

ULA LÓGICA

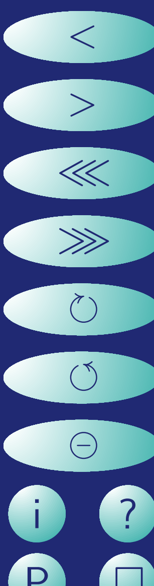


Árbol-SLD = Un árbol de búsqueda

Definición lm3-11: Un árbol-SLD para una cláusula objetivo O_o a partir de T usando la función de selección F es un árbol, cada nodo del cual es una cláusula objetivo, definida como sigue :

- El nodo raíz es O_o.
- Si O es un nodo en el árbol-SLD entonces, para cada cláusula definida Φ en T cuya cabeza se puede unificar con F(O), el nodo O tiene un hijo que es un resolvente de O con una variante de Φ.

ULA LÓGICA



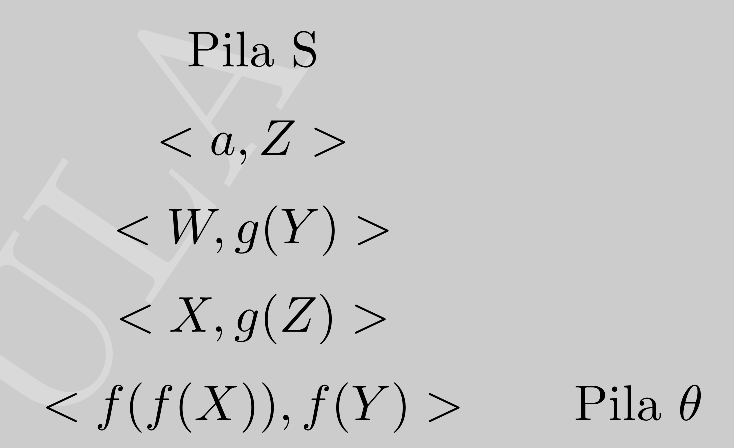
El algoritmo de Unificación.

> Unificación es un método (algoritmo) para establecer si dos términos son "sintácticamente" iguales. Unificar significa, precisamente, hacer que los dos términos sean iguales (aplicando substituciones y reemplazos ó, como lo llamaremos ahora, ligas o enlaces (bindings)).

Suponga que queremos unificar

$$p(a, W, X, f(f(X))) \text{ y } p(Z, g(Y), g(Z), f(Y))$$

Comencemos colocando cada par de términos correspondiente en una pila:



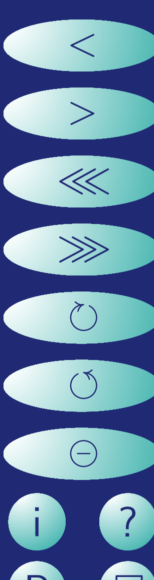
Pila S

$$< a, Z >$$

$$< W, g(Y) >$$

$$< X, g(Z) >$$

$$< f(f(X)), f(Y) > \qquad \text{Pila } \theta$$



El proceso de unificación consiste de remover los pares de la pila S y colocar los enlaces correspondientes en la pila θ de acuerdo al siguiente algoritmo.

Repeat

if S está vacía **then** termine y devuelva θ (el unificador más general)

else

begin

remueva el siguiente par de términos $< s_1, s_2 >$ de S

y construya $< e_1, e_2 >=< s_1\theta, s_2\theta >$;

if e_1 y e_2 son constantes distintas

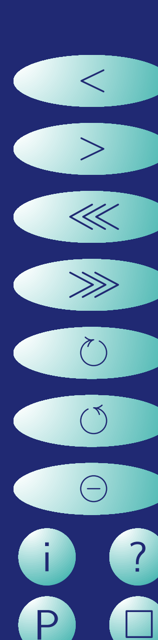
then escriba FALLA y termine.

else if e_1 y e_2 son términos funcionales con nombres de función distintos

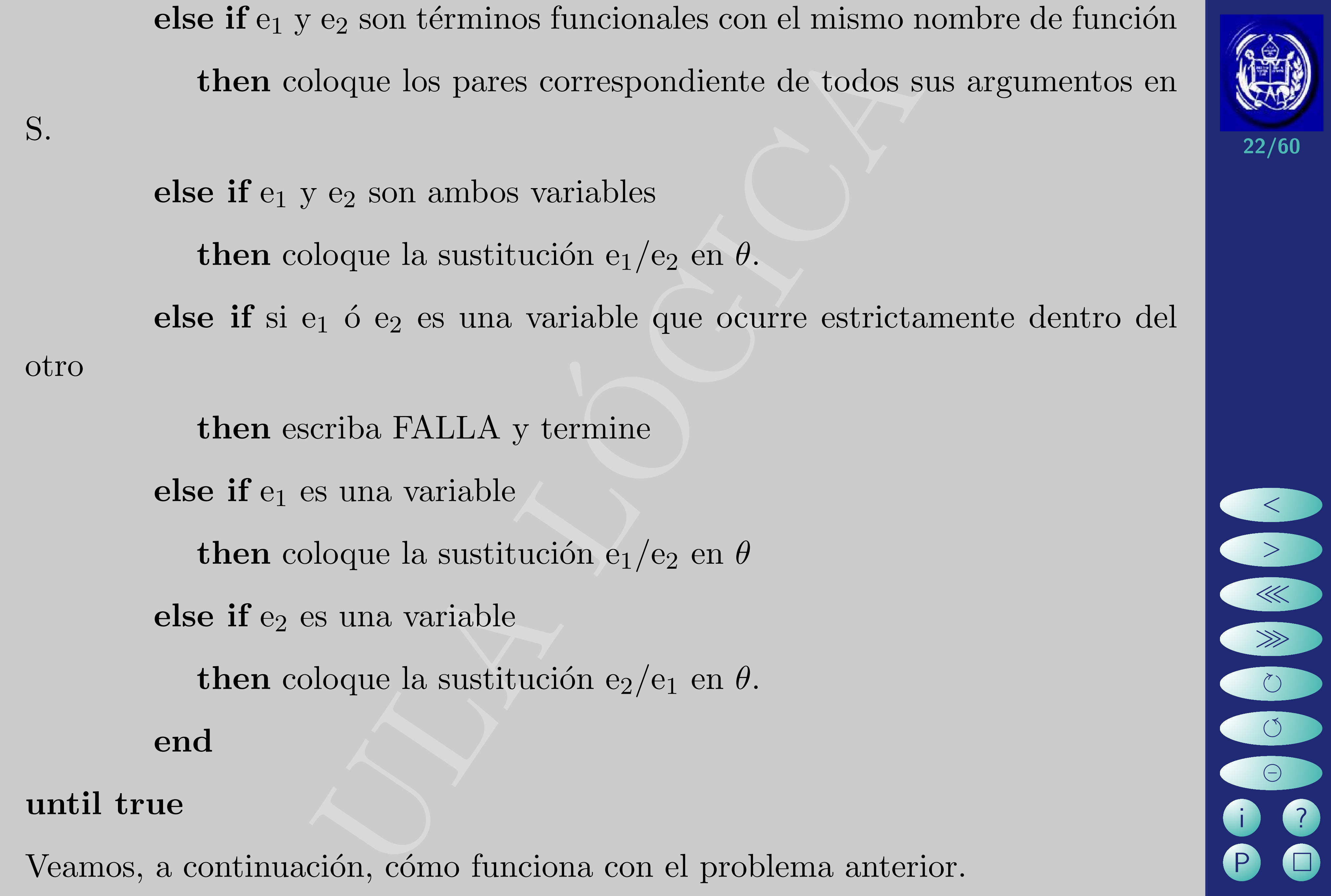
then escriba FALLA y termine.

else if entre e_1 y e_2, uno es una constante y el otro es una función

then escriba FALLA y termine.



ULA LÓGICA



else if e_1 y e_2 son términos funcionales con el mismo nombre de función

then coloque los pares correspondiente de todos sus argumentos en S.

else if e_1 y e_2 son ambos variables

then coloque la sustitución e_1/e_2 en θ.

else if si e_1 ó e_2 es una variable que ocurre estrictamente dentro del otro

then escriba FALLA y termine

else if e_1 es una variable

then coloque la sustitución e_1/e_2 en θ

else if e_2 es una variable

then coloque la sustitución e_2/e_1 en θ.

end

until true

Veamos, a continuación, cómo funciona con el problema anterior.

ULA LÓGICA

Así funciona el algoritmo de unificación:

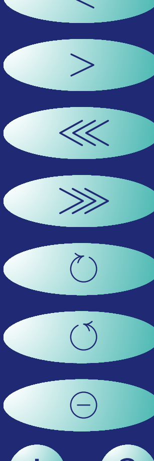
Paso 1 Remueva $< a, Z >$ de la pila S, construya

$$< e1, e2 >=< a\{\}, Z\{\}>$$

y coloque la sustitución correspondiente en la pila del unificador.

Pila S	Pila θ
$< W, g(Y) >$	
$< X, g(Z) >$	
$< f(f(X)), f(Y) >$	Z/a

ULA LÓGICA



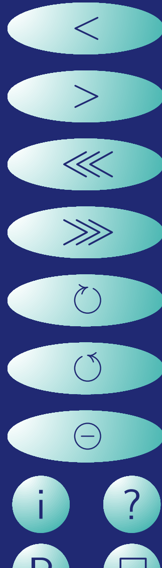
Paso 2 Remueva $< W, g(Y) >$ de S, construya

$$< e1, e2 > = < W\{Z/a\}, g(Y)\{Z/a\} >$$

y coloque la sustitución en la pila del unificador θ.

Pila S	Pila θ
$< X, g(Z) >$	$W/g(Y)$
$< f(f(X)), f(Y) >$	Z/a

ULA LÓGICA



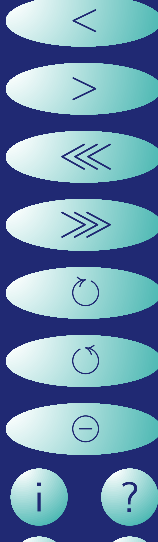
Paso 3 Remueva $< X, g(Z) >$ de S, construya

$$< e1, e2 >=< X\{Z/a, W/g(Y)\}, g(Z)\{Z/a, W/g(Y)\} >$$

y actualice θ.

Pila S	Pila θ
	$X/g(a)$
	$W/g(Y)$
$< f(f(X)), f(Y) >$	Z/a

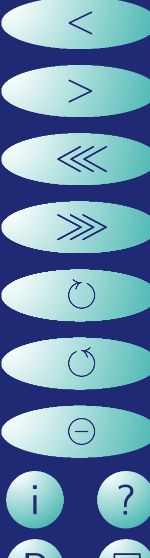
ULA LÓGICA



Paso 4 Remueva $< f(f(X)), f(Y) >$, construya $< e1, e2 >$ y, dado que estamos comparando dos funciones con el mismo nombre, coloque los pares correspondientes de los argumentos en S:

Pila S	Pila θ
	$X/g(a)$
	$W/g(Y)$
$< f(g(a)), Y >$	Z/a

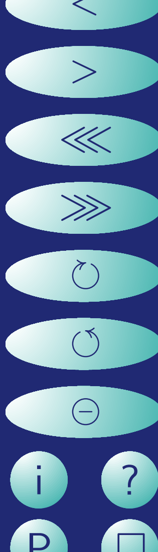
ULA LÓGICA



Paso 5 Remueva $< f(g(a)), Y >$ y complete el unificador:

Pila S	Pila θ
	$Y/f(g(a))$
	$X/g(a)$
	$W/g(Y)$
	Z/a

ULA LÓGICA



Hay un detalle importante en el algoritmo de UNIFICACION. Se le conoce como **el chequeo de ocurrencia**.

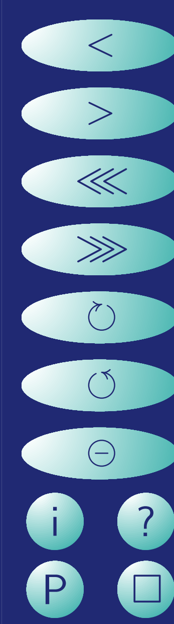
Considere el programa y la pregunta siguiente:

C1: $igual(W, W)$. Unica cláusula del programa.

Q1: $?igual(s(X), s(s(X)))$. La pregunta.

Actividad: Revise el algoritmo de unificacion y responda: ¿Cómo se comporta el algoritmo de unificación si tratamos de probar esa pregunta con esa cláusula?.

¿Qué le ocurre al algoritmo si desconectamos el chequeo de ocurrencia?.



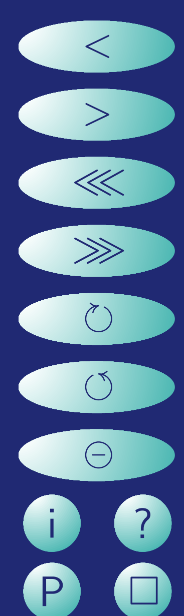
ULA LÓGICA

La interpretación procedimental de los programas lógicos

Esta es la idea que ***funda*** el campo de la **programación lógica** y, luego, en los ultimos años, originó la **lógica computacional**: *Podemos interpretar las fórmulas lógicas como código para el computador*, como se muestra a continuación.

Actividad: Verifique esta afirmación sobre un código PROLOG

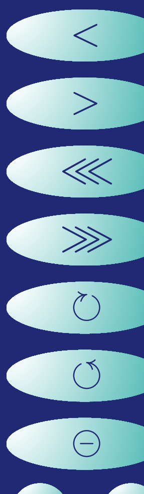
ULA LÓGICA



La interpretación (en el sentido de “leer cómo”)

- El programa es *invocado* cuando se presenta una cláusula objetivo, digamos O, al sistema.
- La *ejecución* de un programa es el proceso de encontrar una refutación SLD de O a partir del programa T
- $B \leftarrow A_1 \wedge .... \wedge A_n$ $(n > 0)$ se interpreta como una declaración de procedimiento. B es el nombre del procedimiento. El cuerpo $\{A_1,....,A_n\}$ de procedimiento consiste de llamadas a procedimientos (cada A_i es una llamada).
- $B \leftarrow$ es la afirmación de un hecho.
- $\square$ la cláusula nula se interpreta como una instrucción de terminación.

ULA LÓGICA



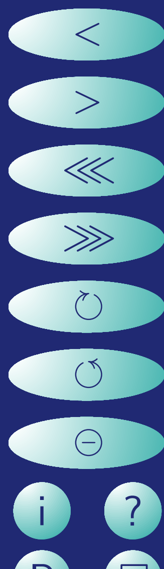
Las estructuras tradicionales de programación

Las estructuras tradicionales pueden ser simuladas así:

- Una instrucción condicional puede representarse con varias cláusulas definidas para el mismo predicado.
- Un ciclo es un procedimiento recursivo.
- Una estructura de datos es un término.

Atentos a los ejemplos por correo.

ULA LÓGICA



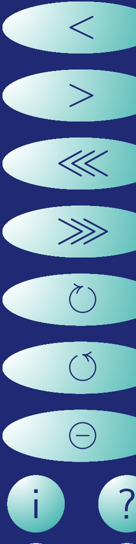
Más eficiencia en la programación lógica: podando el árbol SLD con el CUT

El operador ! es un recurso de CONTROL en el lenguaje PROLOG que queremos discutir para ilustrar algunas consideraciones de eficiencia computacional en el paradigma basado en lógica.

Actividad: Consulte el manual de PROLOG y averigüe cómo funciona el CUT (!).

Actividad: Pruebe el siguente ejemplo para confirmar esa información.

Sea la relación $minimo(X, Y, Z)$ **tal que si X es mayor o igual a Y entonces Z = Y de lo contrario Z = X. A continuación un programa PROLOG que implementa la relación (PRUEBELO):**



ULA LÓGICA

$$mínimo(X\ ,\ Y\ ,\ Z)\ si\ mayor_o_igual(X\ ,Y) \wedge Z = Y$$

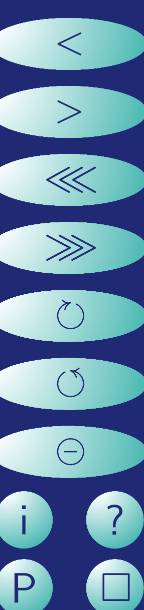
$$mínimo(X\ ,\ Y\ ,\ Z)\ si\ Y > X \wedge Z = X$$

$$mayor_o_igual(X\ ,\ Y)\ si\ X > Y$$

$$mayor_o_igual(X\ ,\ Y)\ si\ X = Y$$

El árbol de resolución para ese programa es el siguiente. Observe como se duplican algunos elementos.

ULA LÓGICA



$?minimo(3,2,Z)$

$\downarrow \qquad\qquad \searrow$

$?mayor_o_igual(3,2) \wedge Z = 2 \qquad\qquad ?2 > 3 \wedge Z = 3$

$\downarrow \qquad\qquad \searrow \qquad\qquad \searrow$

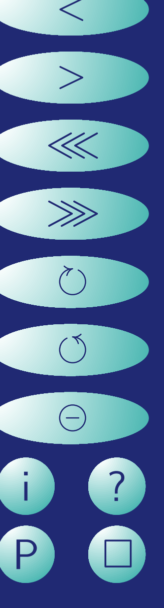
$?3 > 2 \wedge Z = 2 \qquad ?3 = 2 \wedge Z = 2 \qquad \Box$

$\downarrow \qquad\qquad \downarrow$

$\Box \qquad\qquad \Box$

Una forma de eliminar esa redundancia de cálculo (si no la ha visto, pregunte al profesor) es agregando el cut (!) en una de las cláusulas como se muestra a continuación.

ULA LÓGICA



$$minimo(X\ ,\ Y\ ,\ Z) \leftarrow mayor_o_igual(X\ ,\ Y) \wedge ! \wedge Z = Y$$

Una vez que se ha establecido que X es mayor o igual que Y (es decir, al cruzar por el cut. En el ejemplo esto ocurre porque X=3 y Y=2) el sistema “olvida” todas las alternativas para probar *minimo* (es decir, no guardará la segunda cláusula *minimo*) y todas las alternativas para probar los predicados a la izquierda del cut (es decir, la otra cláusula para *mayor_o_igual* también será ignorada).

En términos del árbol SLD, esto significa cortar las ramas a la derecha (vea cuales) y, en términos computacionales, eso a su vez significa un ahorro de memoria y tiempo de computación.

Note, sin embargo, que el verdadero problema es que la restricción: $\neg X > Y \vee \neg X = Y$ no se puede **expresar** en el lenguaje y la plataforma actual (No tenemos donde colocarla).

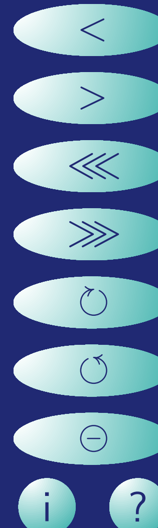
ULA LÓGICA

Evaluando a resolución SLD como paradigma de programación

El paradigma de programación consiste, por ahora, de las siguientes convenciones:

- El lenguaje es la lógica de las cláusulas de Horn.
- Un programa en un conjunto de cláusulas definidas.
- Una pregunta es una cláusula negativa.
- Una computación es una derivación SLD.
- Una ejecución es una búsqueda exhaustiva sobre el árbol SLD hasta encontrar $\square$.

ULA LÓGICA



¿Cuáles problemas se pueden representar con el lenguaje de este paradigma?

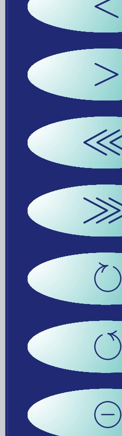
Cualquier función recursiva parcial puede representarse como un programa de cláusulas definidas .

$$\langle \mathrm{X}_1 \;, \mathrm{X}_2 \;, \ldots\ldots, \mathrm{X}_\mathrm{n} \;, \mathrm{X}_{\mathrm{n}+1} \rangle \in \mathrm{Rp} \leftrightarrow \mathrm{f}(\mathrm{X}_1 \;, \ldots, \mathrm{X}_\mathrm{n}) = \mathrm{X}_{n+1}$$

Es decir, piense que reescribimos la funcion suma $\mathrm{X} = \mathrm{Y} + \mathrm{Z}$, como una relación con el predicado suma(X,Y,Z). Nada se pierde. Lo mismo ocurrirá con cualquier función. Los matemáticos suelen asociar problemas con funciones (que los "modelan") y, por ello, respondemos a esta pregunta identificando el tipo de funciones que puede representar nuestro paradigma.

¿Dejamos algo fuera?.

ULA LÓGICA



¿Qué problemas puede resolver el paradigma?

(La clase de las f.r.p. $\equiv$ funciones Turing-computables)

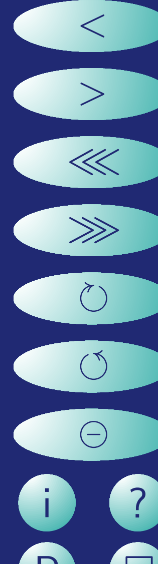
Esto significa, *todos* los problemas efectivamente solucionables.

Es difícil resumir el porqué, pero esta respuesta está conectada con la lámina anterior. La teoría de la computación gira en torno a esas funciones Turing-computables o computables por una máquina de Turing.

Hablaremos de esto hacia el final del curso, pero:

Actividad: Precise qué es una máquina de Turing.

ULA LÓGICA

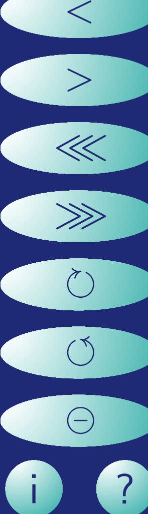


¿Es un lenguaje conveniente para expresar problemas?

Eso depende de con qué lo estemos comparando:

- Lógica vs. código imperativo: son completamente equivalentes (¿Qué los distingue, entonces?).
- Lógica vs. código orientado a objetos: No tenemos clases en lógica, pero el resto de las estructuras (atributos y métodos) es representable. Las clases se pueden definir en metalógica o con lógica modal. Lo mismo ocurre con los agentes.
- Formal clausal vs. Lógica de primer orden: La forma clausal es menos expresiva. De alliel problema del CUT.
- Cláusulas de Horn vs. forma clausal: Las cláusulas de Horn son todavías más limitadas en expresividad, pero tienen virtudes computacionales sobresalientes.

ULA LÓGICA



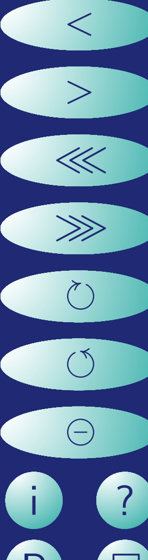
¿Es SLD un método sano?

Sano (*sound* en Inglés) quiere decir CORRECTO. Es decir, que no va a producir soluciones equivocadas. La respuesta es:

Si, siempre y cuando se incluya el chequeo de ocurrencia, pero este cuesta caro. O costaba, con las modernas velocidades de computación se vuelve cada vez más asequible.

Hablaremos luego sobre sanidad (y completitud).

ULA LÓGICA

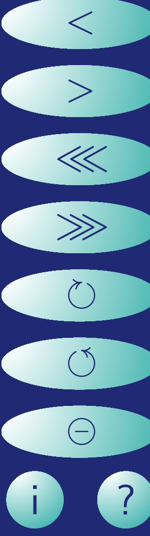


¿Es SLD un método completo?

- Para las cláusulas de Horn, resolución (y resolución SLD) es completa enmodo de refutación .
- Para las cláusulas de Horn, resolución (y resolución SLD) es completa en modo de afirmación *si* la pregunta es atómica o una conjunción de átomos.
- Para cláusulas generales, resolución es completa para refutación si se le completa con *factorización.*
- Para cláusulas generales, resolución *no* es completa en modo de afirmación, tenga o no tenga factorización.

Noten que en todo esto estamos hablando de la regla de resolución SIN considerar estrategias de búsqueda.

ULA LÓGICA



APARTADO: Complementos posibles para resolución

Resolución, para lógica de predicado, debe ser complementada con otra regla de inferencia. De otra forma, no sería completa como regla de inferencia para lógica de predicados.

Esa regla complementaria es

factorización.

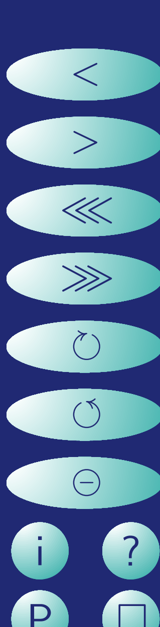
$L1 \vee L2 \vee otros = (L1 \vee otros)\theta$ si L1 y L2 tienen un u.m.g θ

Pero cuidado!!.. Pueden surgir combinaciones "embarazosas".

$ama_a(sra_thatcher, X) \leftarrow respalda(X, Y) \wedge \neg\, ama_a(Y, socialismo)$

$ama_a(sra_thatcher, socialismo) \leftarrow respalda(socialismo, sra_thatcher)$

ULA LÓGICA



¿Qué queda fuera del alcance de la programación lógica?

La lógica de primer orden es semi-decidible → el lenguaje de cláusulas es semi-decidible. Esto significa que:

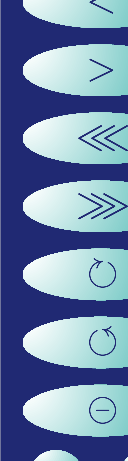
"No podemos construir un interpretador completo de programas lógicos para el que se pueda garantizar que no caerá en ciclos ante un programa y una pregunta cualesquiera".(1).

"Dada una conjunción de un conjunto de cláusulas definidas T y una conjunción de átomos, Φ , *HAY* algoritmos capaces de determinar en tiempo finito si existe una refutación SLD de ← Φ a partir de T en tiempo finito, si una existe ".

SLD + una estrategia de selección justa, es un ejemplo.

(1) *PERO ESTO AFECTA A CUALQUIER FORMALISMO COMPUTACIONAL.*

ULA LÓGICA



De regreso a la teoría de modelos

Recordemos los discutido sobre interpretaciones y modelos en lógica con un ejemplo.

Sea P el siguiente, muy sencillo, programa lógico:

$entiende(bob, X) \leftarrow entiende(X,\text{logica}).$

$entiende(jacinto,\text{logica}).$

que tiene por G(P) (forma básica):

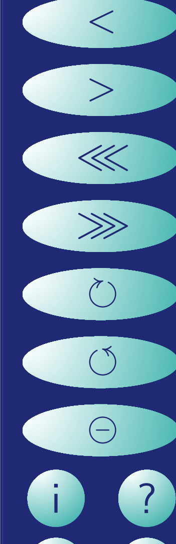
$entiende(bob, bob) \leftarrow entiende(bob,\text{logica}).$

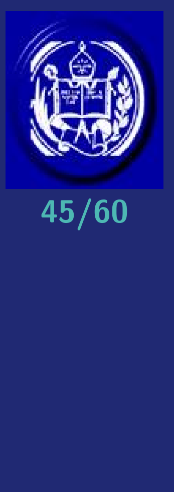
$entiende(bob, jacinto) \leftarrow entiende(jacinto, \text{logica}).$

$entiende(bob,\text{logica}) \leftarrow entiende(\text{logica, logica}).$

$entiende(jacinto,\text{logica}).$

ULA LOGICA





Podemos abreviar ese programa así:

$bb \leftarrow bl.$

$bj \leftarrow jl.$

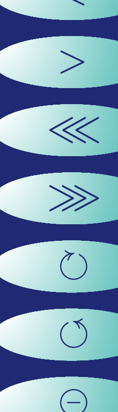
$bl \leftarrow ll.$

$jl.$

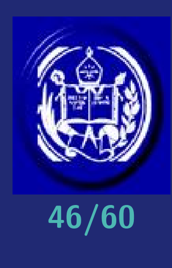
Considere los siguientes **conjuntos de átomos** IH_1 e IH_2.

$IH_1 = \{bb, bl\}$ es una interpretación de Herbrand para P.

$IH_2 = \{jl, bj\}$ también.



ULA LÓGICA



Note, sin embargo, que $IH_2 = \{jl, bj\}$ hace que G(P) se pueda re-escribir así:

falso ← **falso**

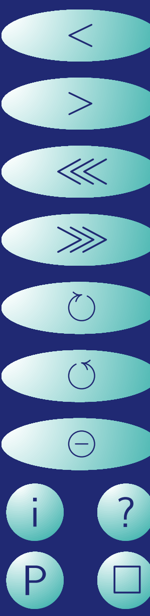
cierto ← **cierto**

falso ← **falso**

cierto

es decir, los átomos en ese conjunto SATISFACEN (hacen ciertas) todas las fórmulas (cláusulas) de P y por lo tanto, IH_2 es también un modelo de Herbrand (modelo-H) de P:

ULA LÓGICA



Este es un resultado importante:

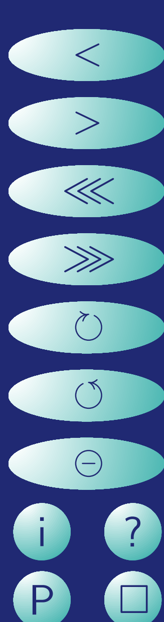
Un conjunto de cláusulas P tiene un modelo de algún tipo

..... *si y solo si* P tiene un modelo-H

..... *si y solo si* G(P) tiene un modelo-H

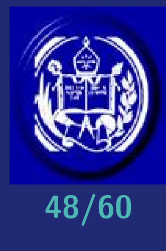
Porque significa que sólo con manipulación simbólica de una representación sin variables es posible encontrar una interpretación que satisfaga la formulación, si existe.

¿Y.. por qué eso es tan importante?.



ULA LÓGICA

La látis de modelos de un programa definido



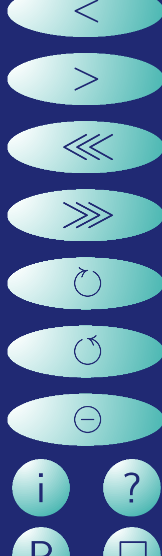
Estamos interesados en un conjunto de átomos que satisfaga al programa y que sea mínimo.

Este conjunto será considerado como *el significado* del programa.

Teorema lm3-4: Sea $M(P) = \{M1, M2, ...\}$ el conjunto de todos los modelos de un programa definido P y sea $\subseteq$ la relación de orden parcial asociada a $M(P)$. El copo $(M(P), \subseteq)$ forma necesariamente una látis completa y en consecuencia tiene un único modelo mínimo.

Recuerden lo visto en la sección **latises para qué** de la unidad 2.

ULA LÓGICA



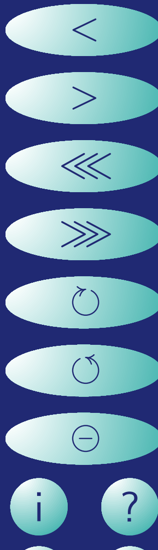
ULA LÓGICA

Actividad: Ubique artículos de programación lógica escritos por Robert Kowalski y VanEmden después de 1972, en alguna biblioteca cercana o en Internet.

La base de Herbrand es siempre un modelo-H (el más grande) de $G(P)$.

En general, los programas que no son definidos (son indefinidos) no producen una látis completa y pueden tener varios modelos mínimos. Al no tener un sólo modelo mínimo es más difícil asignarles un significado no ambiguo.

Por esta razón TODA la teoría que revisamos hasta esta unidad 3 de este curso, se basa en programas lógicos definidos.



Modelos mínimos y consecuencia lógica

Si P es un programa definido y $MM(P)$ su modelo mínimo y q un átomo básico cualquiera:

$$P \models q \leftrightarrow q \in MM(P)$$

Teorema lm3-5: La intersección I^* de todos los modelos de un programa definido P es el modelo mínimo $MM(P)$ de P.

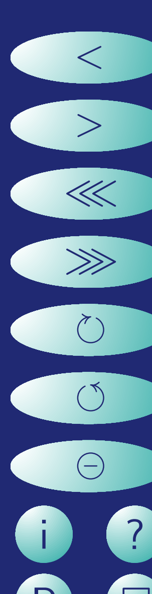
Prueba lm3-5: Supongamos que I^* no es el modelo mínimo:

.... entonces debe existir algún modelo M_j tal que $M_j \subset I^*$

.... entonces debe existir algún átomo q tal que $q \notin M_j$ y $q \in I^*$

.... pero $q \in I^*$ implica que $q \in M_i$ **para todo i**, incluyendo al "j" mencionado antes, contradiciendo a $q \notin M_j$. Por tanto, la suposición inicial debe ser falsa.

ULA LÓGICA



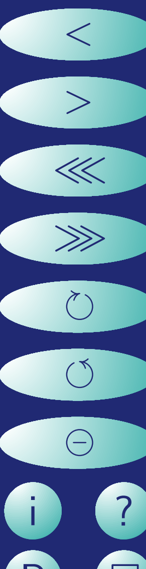
Teorema lm3-6 [Skolem-Herbrand-Gödel]: Un conjunto de cláusulas S es tal que no-puede-ser-satisfecho si y solo si existe algún subconjunto de G(S) que no tiene modelos-H.

Este teorema es un complemento al principio de deducción (que vimos en unidad 1) y nos permite hacer reducción al absurdo sobre cláusulas.

Las siguientes afirmaciones son todas equivalentes:

$$
\begin{aligned}
q \in MM(P) \quad &\leftrightarrow \quad P \models q \\
&\leftrightarrow \quad P \cup \{\neg q\} \text{ no puede ser satisfecho.} \\
&\leftrightarrow \quad \text{Algún subconjunto finito de } G(P \cup \{\neg q\}) \\
&\qquad\quad \text{no tiene modelos-H.} \\
&\leftrightarrow \quad \text{Existe algún subconjunto finito } g \subseteq G(P) \\
&\qquad\quad \text{tal que } g \cup \{\neg q\} \text{ no tiene modelos-H.} \\
&\leftrightarrow \quad \text{Algún subconjunto finito de } G(P) \text{ implica} \\
&\qquad\quad \text{a } q.
\end{aligned}
$$

ULA LÓGICA



Modelos mínimos y derivabilidad

Las siguientes afirmaciones muestran la relación entre los elementos del modelo mínimo y las derivaciones SLD.

(P es un programa definido)

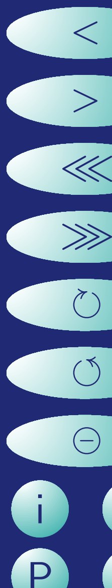
$$\forall q \in B(P), \quad \begin{array}{rcl} q \in MM(P) & \leftrightarrow & P \models q \\ & \leftrightarrow & P \cup \{\neg q\} \text{ no puede ser satisfecho.} \\ & \leftrightarrow & P \cup \{\neg q\} \vdash_{SLD} \Box \end{array}$$

Gracias a la correctitud y completitud de Resolución SLD uno puede decir lo siguiente acerca del conjunto de éxito ($SS(P)$) de P:

$$SS(P) = MM(P)$$

Es decir, cualquier átomo con el cual resolución SLD tenga exito, es parte del modelo mínimo y viceversa.

ULA. LÓGICA



Construcción progresiva de modelos mínimos

(P **sigue siendo** un programa definido).

Siguiendo con lo dicho al final de la lámina anterior, si tuvieramos otra forma de calcular el modelo mínimo, tendriamos una forma de verificar si resolución SLD es sana (y completa). Pues, tenemos otra forma:

Considere $H = \{jacinto, gandhi\}$ y P:

$noble(X) \leftarrow ama(X).$

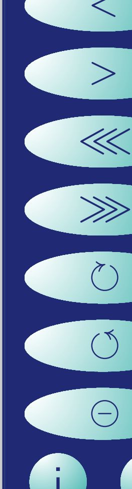
$ama(gandhi).$

De nuevo, $G(P)$ se puede escribir en forma compacta como:

$ng \leftarrow ag.$

$nj \leftarrow aj.$

$ag.$



ULA. LÓGICA

Sea

$$I_1 = \{ng, aj\},$$

por el método descrito en el texto obtenemos:

$$I_2 = \{nj, ag\},$$

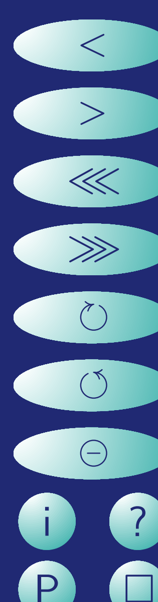
y seguidamente:

$$I_3 = \{ng, ag\} = I_4 = I_5 = \ldots.$$

I_3 es un **punto fijo** y, además, es igual a $MM(P)$..!!!

Nota: Si estamos buscando $MM(P)$ y comenzamos con la interpretación vacía ($\emptyset$) podemos estar seguros de que arribaremos a $MM(P)$.

ULA LÓGICA



Interpretación de punto fijo del modelo mínimo

$$Tp(I_k) = I_{k+1}$$

donde $Tp : \mathcal{P}(B(P)) \rightarrow \mathcal{P}(B(P))$ se define como:

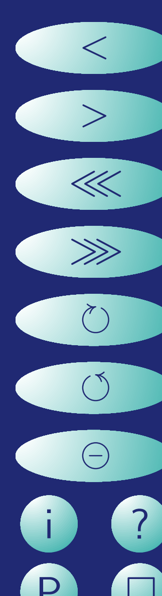
$$Tp(I) = \{q | (q \leftarrow cuerpo) \in G(P) \text{ y } cuerpo \text{ es } \textbf{cierto} \text{ en I}\}$$

Para armar una definición del significado de P es conveniente que Tp sea una función monótona y continua.

Tp es nuestra función para construir significados ¿Por qué?.

Actividad: Responda la pregunta.

ULA LÓGICA



APARTADO: Sobre funciones, copos y puntos fijos

Una función f en un copo $(S, \preceq)$ es monótona si:

$$\forall u \in S, v \in S, f(u) \preceq f(v) \leftarrow u \preceq v$$

y continua si

Para todo subconjunto s de S:

$$f(mcs(s)) = mcs\{f(u)|u \in s\}$$

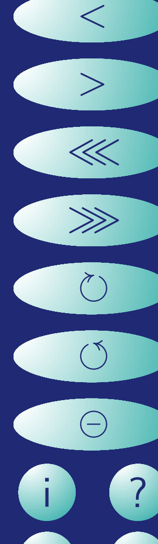
Para una función f sobre $(S, \preceq)$, un elemento dado $u \in S$ es un punto fijo de f si y sólo si:

$$f(u) = u$$

un elemento u$\in S$ es un pre-punto fijo de f si y solo si:

$$f(u) \preceq u$$

ULA LÓGICA



Teorema lm3-7 [Knaster-Tarski]: Toda función monótona f sobre un látis completo $(S, \preceq)$ tiene un punto fijo mínimo en S que es también el pre-punto fijo mínimo en S.

Teorema lm3-8 [Primer teorema de recursión de Kleene]: Sea N el conjunto de los números naturales {0,1,2...} y sea ω su cardinalidad. Toda función f continua sobre un látis completo $(S, \preceq)$ cuyo elemento base (mínimo) es $\perp$ tiene un punto fijo mínimo en S dado por:

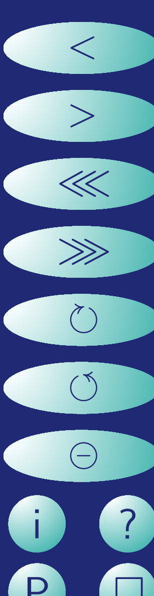
$$f \uparrow \omega = mcs(f^k(\perp)|k \in N\}$$

$$\text{y si } \preceq \text{ es } \subseteq\text{: } f \uparrow \omega = mcs(f^k(\emptyset)|k \in N\} = \cup_{k \in N} f^k(\emptyset)$$

si $f = Tp$ sobre el látis de interpretaciones de un programa definido P, entonces $f \uparrow \omega$ (f a la omega) es **exactamente** el modelo mínimo de P.

Ya sabemos como obtener el significado de P!!

ULA. LÓGICA



Semántica denotacional de programas definidos

¿Qué queremos decir con "semántica"?.

RESPUESTA: Queremos atribuirle a cada nombre de predicado p en un programa P, una relación r bien definida.

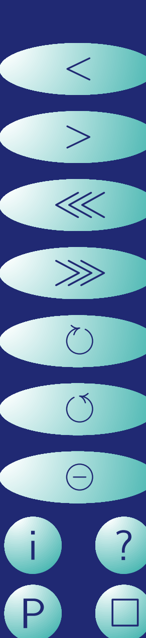
r debe ser mínima para evitar redundancia y única para evitar ambigüedad!

Semántica basada en consecuencia lógica.

$$\{t|P \models p(t)\}$$

Observen que, con esta definición de semántica, el "significado" de este programa P es el conjunto formado por todos los elementos de una interpretación, exactamente en el sentido que discutimos en unidad 1.

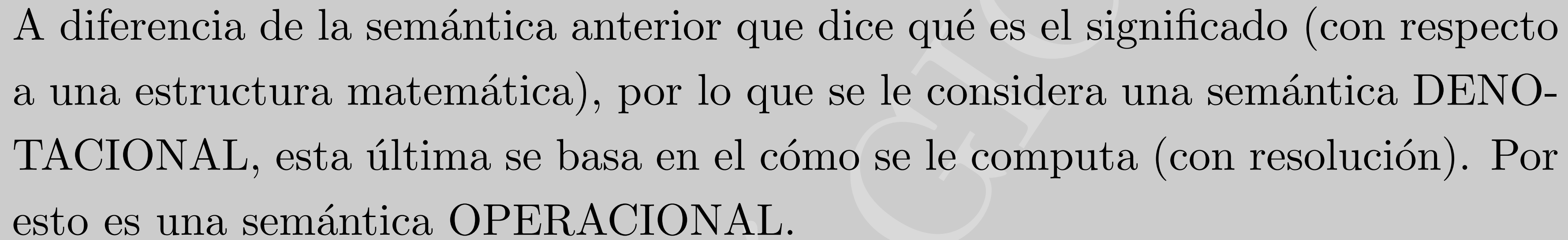
Esa interpretación, desde luego, no es cualquiera. Ya vimos que es el modelo más pequeõ que, siempre existe y es único para los programas definidos.



ULA LÓGICA

Semántica operacional

$$\{t|P \cup \{?p(t)\} \vdash_{SLD} \Box\} = \{t|p(t) \in SS(P)\}$$



A diferencia de la semántica anterior que dice qué es el significado (con respecto a una estructura matemática), por lo que se le considera una semántica DENOTACIONAL, esta última se basa en el cómo se le computa (con resolución). Por esto es una semántica OPERACIONAL.

Hablaremos de la relación entre semánticas operacionales y las populares máquinas virtuales en un debate.

Semántica de punto fijo y su relación con las otras

$$\{t|p(t) \in mpf(Tp)\} = \{t|p(t) \in Tp \uparrow \omega\} = \{t|P \cup \{?p(t)\} \vdash_{SLD} \Box\} = \{t|P \models p(t)\}$$

Quizás el resultado más importante el programación lógica es este. Todas las semánticas coinciden (es decir, apuntan a mismo conjunto como significado).

Voila!

Fin de unidad 3

Hemos completado la especificación operacional y la semántica denotacional de un lenguaje de programación.

Este es un ejercicio teórico para fundamentar un lenguaje usando abstracciones matemáticas.

Sugiero que se pregunten para qué sirve todo esto.

Piensen como se sería la evaluación de otro paradigma de programación (como OxO y JAVA) en los términos que hemos usado en esta unidad.

ULA LÓGICA