Programación lógica y Prolog

José A. Alonso Jiménez,
José L. Ruiz Reina y
Francisco J. Martín Mateos

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Progr. lógica relacional: Sintaxis

- **Necesidad de ampliación:**
  - Lenguaje natural:
    * Javier defiende a todos sus jugadores.
    * Finidi es un jugador de Javier.
    * Por tanto, Javier defiende a Finidi.
  - Lógica relacional:
    * $F_1 : (\forall J)[jugador\_de(J, javier) \rightarrow defiende(javier, J)]$
    * $F_2 : jugador\_de(finidi, javier)$
    * $F_3 : defiende(javier, finidi)$
    * $\{F_1, F_2\} \models F_3$
  - Programación lógica relacional:
    * $C_1 : defiende(javier, J) \leftarrow jugador\_de(J, javier).$
    * $C_2 : jugador\_de(finidi, javier) \leftarrow .$
    * $C_3 : defiende(javier, finidi) \leftarrow .$
    * $\{C_1, C_2\} \models C_3$

Progr. lógica relacional: Sintaxis

- **Gramática de la programación lógica relacional:**

```
constante ::= <palabra comenzando con minúscula>
variable  ::= <palabra comenzando con mayúscula>
término   ::= constante | variable
predicado ::= <palabra comenzando con minúscula>
átomo     ::= predicado[(término[,término]*)]
expresión ::= término | átomo
cláusula  ::= cabeza <- cuerpo
cabeza    ::= [átomo]
cuerpo    ::= [átomo[,átomo]*]
cláusula definida ::= átomo <- cuerpo
```

- **Programa lógico relacional**
  - **Def.: Conjunto de cláusulas definidas**
  - **Ejemplo (Prog1)**

```
defiende(javier,J) <- jugador_de(J,javier).
jugador_de(finidi,javier)  <-.
```

- **Expresiones básicas: sin variables**

Progr. lógica relacional: Semántica

- **Universo de Herbrand:**
  - $UH(P)$ = conjunto de términos básicos de P
  - Ejemplo: $UH(Prog1) = \{javier, finidi\}$
- **Base de Herbrand:**
  - $BH(P)$ = conjunto de átomos construidos con predicados de P y términos básicos de P
  - Ejemplo: La BH del programa anterior es:

```
{defiende(javier,javier),   defiende(javier,finidi),
 defiende(finidi,javier),   defiende(finidi,finidi),
 jugador_de(javier,javier), jugador_de(javier,finidi),
 jugador_de(finidi,javier), jugador_de(finidi,finidi)}
```

- **Interpretación de Herbrand:**
  - I interpretación de Herbrand de P si $I \subseteq BH(P)$
  - Ejemplo: Una interpretación de Herbrand de Prog1 es
    `defiende(javier,finidi), jugador_de(finidi,javier)`
  - Nota: Prog1 tiene $2^8 = 256$ interpretaciones de Herbrand

Progr. lógica relacional: Semántica

- **Sustitución:**
  - **Def.: Aplicación de las variables en los términos**
  - **Ej.:** $\sigma = \{J/finidi\}$
- **Aplicación de una sustitución a una cláusula:**
  - $C\sigma$**: sustituir cada variable** X **de** C **por** $\sigma(X)$
  - **Ejemplo:**
    $C = defiende(javier, J1) \leftarrow jugador\_de(J, javier).$
    $\sigma = \{J/finidi\}$
    $C\sigma = defiende(javier, J1) \leftarrow jugador\_de(finidi, javier)$
- **Instancias:**
  - C **es instancia de** D **si existe** σ **tal que** $C = D\sigma$**.**
  - **Instancia básicas.**
  - **Ejemplo: Las instancias básicas de Prog1 son:**

```
defiende(javier,javier) <- jugador_de(javier,javier).
defiende(javier,finidi) <- jugador_de(finidi,javier).
jugador_de(finidi,javier) <-.
```

  - **Programa proposicional correspondiente al básico (Prog1b):**

```
d-j-j  <- jd-j-j.
d-j-f  <- jd-f-j.
jd-f-j <- .
```

Progr. lógica relacional: Semántica

- **Modelos de Herbrand:**
  - $I \models C$ **si lo es de todas sus instancias básicas.**
  - $I \models P$ **si lo es de todas sus cláusulas.**
  - **Ejemplo de modelo de Prog1:**

```
> (modelos-de-Herbrand *prog1b*)
((D-J-F JD-F-J)
 (D-J-J D-J-F JD-F-J)
 (D-J-J JD-J-J D-J-F JD-F-J))
> (menor-modelo-de-Herbrand *prog1b*)
(D-J-F JD-F-J)
```

Progr. lógica relacional: Semántica

- **Semántica de punto fijo**
  - Programa básico
    básico(P) = $\{C\sigma : C \in P$ y $C\sigma$ es básica$\}$
  - Operador de consecuencia inmediata:

    $$T_P(I) = \{A : (A \leftarrow B_1, \ldots, B_n) \in \text{básico}(P) \text{ y } \{B_1, \ldots, B_n\} \subseteq I\}$$

  - Consecuencias:
    $I_0 = \emptyset$
    $I_1 = T_{Prog1}(I_0) = \{jugador\_de(finidi, javier)\}$
    $I_2 = T_{Prog1}(I_1) = I_1 \cup \{defiende(javier, finidi)\}$
    $I_3 = T_{Prog1}(I_2) = I_2$
  - Menor punto fijo:
    $\text{MPF}(Prog1) = \{ jugador\_de(finidi, javier), defiende(javier, finidi)\}$
  - $\text{MPF}(P) = \text{MMH}(P)$

Programación lógica: Sintaxis

- **Necesidad de ampliación (I):**
  - **Lenguaje natural:**
    "Cada persona tiene un padre"
  - **Programación lógica relacional:**

```
es_padre_de(adan,cain).
es_padre_de(adan,abel).
...
```

  - **Lógica (de primer orden):**
    $(\forall \boldsymbol{X})(\exists \boldsymbol{Y})[\text{es\_padre\_de}(\boldsymbol{Y}, \boldsymbol{X})]$
  - **Programación lógica:**

```
es_padre_de(el_padre_de(X),X) <-.
```

- **Necesidad de ampliación (II):**
  - **Lenguaje natural:**
    * **"El cero es un número entero"**
    * **"Todo número entero tiene un siguiente"**
  - **Lógica de primer orden:**
    * $\text{entero}(0)$
    * $(\forall \boldsymbol{X})[\text{entero}(\boldsymbol{X}) \rightarrow (\exists \boldsymbol{Y})[\text{es\_siguiente}(\boldsymbol{Y}, \boldsymbol{X})]]$
  - **Programación lógica:**

```
entero(0) <-.
es_siguiente(s(X),X) <- entero(X).
```

Programación lógica: Sintaxis

- **Gramática de la programación lógica:**

```
functor   ::= <palabra comenzando con minúscula>
variable  ::= <palabra comenzando con mayúscula>
término   ::= variable | functor[(término[,término]*)]
predicado ::= <palabra comenzando con minúscula>
átomo     ::= predicado[(término[,término]*)]
expresión ::= término | átomo
cláusula  ::= cabeza <- cuerpo
cabeza    ::= [átomo]
cuerpo    ::= [átomo[,átomo]*]
cláusula definida ::= átomo <- cuerpo
```

- **Programa lógico**
  - **Def.: Conjunto de cláusulas definidas**
  - **Ejemplo (Prog2)**

```
suma(0,X,X) <-.
suma(s(X),Y,s(Z)) <- suma(X,Y,Z).
```

  - **Significado**
    $(\forall \boldsymbol{X})[\boldsymbol{suma}(\mathbf{0}, \boldsymbol{X}, \boldsymbol{X})]$
    $(\forall \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})[\boldsymbol{suma}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) \rightarrow \boldsymbol{suma}(\boldsymbol{s}(\boldsymbol{X}), \boldsymbol{Y}, \boldsymbol{s}(\boldsymbol{Z}))]$

Programación lógica: Semántica

- **Universo de Herbrand:**
  - $UH(P)$ = conjunto de términos básicos de P
  - Ejemplo: $UH(Prog2) = \{0, s(0), s(s(0)), \ldots\}$
- **Base de Herbrand:**
  - $BH(P)$ = conjunto de átomos construidos con predicados de P y términos básicos de P
  - Ejemplo: La BH del programa anterior es:

```
{suma(0,0,0), suma(0,0,s(0)), suma(0,0,s(s(0))), ...
 suma(0,s(0),0), suma(0,s(0),s(0)), ...}
```

- **Interpretaciones de Herbrand**
  - I interpretación de Herbrand de P si $I \subseteq BH(P)$
- **Modelos de Herbrand**
  - $I \models C$ si lo es de todas sus instancias básicas
  - $I \models P$ si lo es de todas sus cláusulas
  - Ejemplos de modelos del Prog2:

```
I1 = BH(Prog2)
I2 = {suma(0,0,0), suma(0,s(0),s(0)), ...
      suma(s(0),0,s(0)), suma(s(0),s(0),s(s(0))), ...}
```

  - Menor modelo de Herbrand de P

Programación lógica: Semántica

- **Semántica de punto fijo**
  - Programa básico
    básico(P) = $\{C\sigma : C \in P$ y $C\sigma$ es básica$\}$
  - Operador de consecuencia inmediata:

$$T_P(I) = \{A : (A \leftarrow B_1, \ldots, B_n) \in \text{básico}(P) \text{ y } \{B_1, \ldots, B_n\} \subseteq I\}$$

  - Consecuencias:

$$\begin{aligned}
I_0 &= \emptyset \\
I_1 &= T_{Prog2}(I_0) = I_0 \cup \{ suma(0,0,0), \\
&\qquad suma(0,s(0),s(0)), \ldots\} \\
I_2 &= T_{Prog1}(I_1) = I_1 \cup \{ suma(s(0),0,s(0)), \\
&\qquad suma(s(0),s(0),s(s(0))), \ldots\} \\
I_3 &= T_{Prog1}(I_2) = I_2 \cup \{ suma(s(s(0)),0,s(s(0))), \\
&\qquad suma(s(s(0)),s(0),s(s(s(0)))), \ldots
\end{aligned}$$

  - Menor punto fijo:

$$\begin{aligned}
\text{MPF}(Prog1) = \{ & suma(0,0,0), \\
& suma(0,s(0),s(0)), \ldots, \\
& suma(s(0),0,s(0)), \\
& suma(s(0),s(0),s(s(0))), \ldots\}
\end{aligned}$$

  - MPF(P) = MMH(P)

Unificación

- **Notaciones:**
  - Variables: $X, X_1, X_2, \ldots, Y, Y_1, Y_2, \ldots$
  - Constantes: $a, a_1, a_2, \ldots, b, b_1, b_2, \ldots$
  - Símbolos de función: $f, f_1, f_2, \ldots, g, g_1, g_2, \ldots$
  - Símbolos de predicados: $p, p_1, p_2, \ldots, q, q_1, q_2, \ldots$
  - Términos: $t, t_1, t_2, \ldots, s, s_1, s_2, \ldots$
  - Fórmulas atómicas: $A, A_1, A_2, \ldots, B, B_1, B_2, \ldots$
  - Expresiones: $E, E_1, E_2, \ldots$
- **Representación de variables en Lisp**
  - Variables: `?x, ?x1, ?var`
  - Procedimiento:

```
;;; (es-variable 'x2)   =>  NIL
;;; (es-variable '?x2)  =>  T
(defun es-variable (e)
  (and (symbolp e)
       (equal (char (symbol-name e) 0) #\?)))
```

Unificación

- **Concepto de sustitución**
  - Def.: $\sigma = \{X_1/t_1, \ldots, X_n/t_n\}, n \geq 0$
  - Ejemplos:
    $\{X/a, Z/f(X,Y)\}$
    $\{X/a, Z/f(a,c)\}$
    $\epsilon = \{\}$
  - Ejemplos en Lisp:

```
((?x . a) (?z . (f ?x ?y)))
((?x . a) (?z . (f a c)))
()
```

- **Dominio de una sustitución**
  - Def.: dominio$(\{X_1/t_1, \ldots, X_n/t_n\}) = \{X_1, \ldots, X_n\}$
  - Ejemplo: dominio$(\{X/a, Z/f(X,Y)\}) = \{X, Z\}$
  - Procedimiento

```
;;; (dominio '((?x . a) (?z . (f ?x ?y))))
;;; =>  (?X ?Z)
(defun dominio (s)
  (mapcar #'(lambda (p) (first p))
          s))
```

Unificación

- **Aplicación de una sustitución a una expresión**
  - **Definición:** $\sigma = \{X_1/t_1, \ldots, X_n/t_n\}$

    $X_i\sigma = t_i$

    $X\sigma = X$, si $X \notin$ dominio(σ)

    $f(s_1, \ldots, s_m)\sigma = f(s_1\sigma, \ldots, s_m\sigma)$

    $p(s_1, \ldots, s_m)\sigma = p(s_1\sigma, \ldots, s_m\sigma)$

    $(A \leftarrow B_1, \ldots, B_m)\sigma = A\sigma \leftarrow B_1\sigma, \ldots, B_m\sigma$

    $\{E_1, \ldots, E_n\}\sigma = \{E_1\sigma, \ldots, E_n\sigma\}$
  - **Ejemplo:**

    $p(f(X,Y),Z)\{X/a, Z/f(X,Y)\} =$

    $= p(f(a,Y), f(X,Y))$
  - **Procedimiento:**

```
;;; (aplica '(p (f ?x ?y) ?z)
;;;         '((?x . a) (?z . (f ?x ?y))))
;;; => (P (F A ?Y) (F ?X ?Y))
(defun aplica (expresion sustitucion)
  (sublis sustitucion expresion))
```

Unificación

- **Composición de sustituciones**
  - **Ejemplo:**

$$
\begin{aligned}
E &= p(f(X,Y),Z)\\
\sigma &= \{X/Y, Y/f(a,Z)\}\\
\theta &= \{Y/X, Z/g(X)\}\\
E\sigma &= p(f(Y,f(a,Z)),Z)\\
(E\sigma)\theta &= p(f(X,f(a,g(X))),g(X))\\
\sigma\theta &= \{Y/f(a,g(X)), Z/g(X)\}\\
E(\sigma\theta) &= p(f(X,f(a,g(X))),g(X))
\end{aligned}
$$

  - **Definición:**

$$
\begin{aligned}
\sigma &= \{X_1/t_1,\ldots,X_n/t_n\}\\
\theta &= \{Y_1/s_1,\ldots,Y_m/s_m\}\\
\sigma\theta &= \{X_i/t_1\theta : i \in [1,n], X_i \neq t_i\theta\} \cup\\
&\quad \{Y_j/s_j : j \in [1,m], Y_j \notin \text{dominio}(\sigma)\}
\end{aligned}
$$

  - **Procedimiento:**

```
(defun composicion (s1 s2)
  (append (remove-if
           #'(lambda (p)(eql (first p) (rest p)))
           (mapcar #'(lambda (p)
                       (cons (first p)
                             (aplica (rest p) s2)))
                   s1))
          (remove-if
           #'(lambda (p)(member (first p) (dominio s1)))
           s2)))
```

Unificación

- Propiedades:
  $E(\sigma\theta) = (E\sigma)\theta$
  $\sigma(\theta\phi) = (\sigma\theta)\phi$
  $\sigma\epsilon = \epsilon\sigma = \sigma$

- Unificador
  - Def.: t_1 y t_2 son unificables $\Longleftrightarrow$ existe σ t.q. $t_1\sigma = t_2\sigma$
  - Def.: σ es un unificador de t_1 y $t_2 \Longleftrightarrow t_1\sigma = t_2\sigma$
  - Ej.: $f(X)$ y $g(Y, Z)$ no son unificables
  - Ej.: Unificadores de $g(g(X))$ y $g(Y)$

Unificador	Instancia
$\{X/3, Y/g(3)\}$	$g(g(3))$
$\{X/f(U), Y/g(f(U))\}$	$g(g(f(U)))$
$\{X/Z, Y/g(Z)\}$	$g(g(Z))$
$\{Y/g(X)\}$	$g(g(X))$

- Comparación de sustituciones
  - σ es más general que $\theta \Longleftrightarrow$ existe ϕ t.q. $\theta = \sigma\phi$

Unificación

- Ej: Comparación de unificadores de $g(g(X))$ y $g(Y)$

$\{X/3, Y/g(3)\}$	$\{Y/g(X)\}\{X/3\}$
$\{X/f(U), Y/g(f(U))\}$	$\{Y/g(X)\}\{X/f(U)\}$
$\{X/Z, Y/g(Z)\}$	$\{Y/g(X)\}\{X/Z\}$
$\{Y/g(X)\}$	$\{Y/g(X)\}\epsilon$

- **Unificador de máxima generalidad**
  - Def.: σ es un unificador de máxima generalidad (UMG) de t_1 y t_2 si
    * σ es un unificador de t_1 y t_2 y
    * σ es más general que cualquier unificador de t_1 y t_2

Unificación

- **Algoritmo de unificación**
  - **Algoritmo**

```
(defun unifica (e1 e2)
  (cond ((equal e1 e2) nil)
        ((es-variable e1)
         (if (ocurre e1 e2) 'fallo
             (list (cons e1 e2))))
        ((es-variable e2)
         (if (ocurre e2 e1) 'fallo
             (list (cons e2 e1))))
        ((or (atom e1) (atom e2)) 'fallo)
        (t (let ((s1 (unifica (first e1) (first e2))))
             (if (eql s1 'fallo)
                 'fallo
                 (let ((s2 (unifica (aplica (rest e1) s1)
                                    (aplica (rest e2) s1)
                   (if (eql s2 'fallo)
                       'fallo
                       (composicion s1 s2)))))))))

(defun ocurre (x e)
  (cond ((null e) nil)
        ((atom e) (eql x e))
        (t (or (ocurre x (first e))
               (ocurre x (rest e))))))
```

Unificación

- **Ej.: Unificación de $f(X, X)$ y $f(Y, a)$**

```
(f X X) (f Y a) {}
  (X X)   (Y a) {X/Y}
    (Y)     (a) {Y/a}
umg = {}{X/Y}{Y/a} = {X/a, Y/a}
```

- **Ej.: Unificación de $f(a, X)$ y $f(b, Y)$**

```
(f a X) (f b Y) {}
  (a X)   (b Y) Fallo
```

- **Ej.: Unificación de $f(X, X)$ y $f(a, b)$**

```
(f X X) (f a b) {}
  (X X)   (a b) {X/a}
    (a)     (b) Fallo
```

- **Ej.: Unificación de $f(Y, X)$ y $f(X, g(Y))$**

```
(f Y X)  (f X (g Y))  {}
  (Y X)    (X (g Y))  {Y/X}
    (X)      ((g X))  Fallo
```

- **Ej.: Unificación de $f(X, X)$ y $f(Y, g(Y))$**

```
(f X X)  (f Y (g Y))  {}
  (X X)    (Y (g Y))  {X/Y}
    (Y)      ((g Y))  Fallo
```

Unificación

- **Ej.: Unificación de**
  $p(U, a, g(U))$ y $p(f(X, Y), X, Z)$

```
(p U a (g U))          (p (f X Y) X Z)   {}
  (U a (g U))            ((f X Y) X Z)   {U/(f X Y)}
    (a (g (f X Y)))              (X Z)   {X/a}
      ((g (f a Y)))                (Z)   {Z/(g (f a Y))}
umg = {U/(f a Y), X/a, Z/(g (f a Y))}
```

- **Ej.: Unificación de**
  $p(U, a, g(U))$ y $p(f(X, Y), X, Y)$

```
(p U a (g U))          (p (f X Y) X Y)   {}
  (U a (g U))            ((f X Y) X Y)   {U/(f X Y)}
    (a (g (f X Y)))              (X Y)   {X/a}
      ((g (f a Y)))                (Y)   Fallo
```

Resolución SLD

- **Ejemplo de resolución SLD**
  - **Programa**

```
defiende(javier,J)      <- jugador_de(J,javier).
defiende(javier,J)      <- jugador_de(J,javier).
jugador_de(X,Y)         <- juega_en(X,E), entrena(Y,E).
juega_en(finidi,betis) <-.
entrena(javier,betis)  <-.
```

  - **Objetivo**

```
<- defiende(javier,finidi).
```

  - **Significado del objetivo:** $\neg defiende(javier, finidi)$
  - **Resolución SLD**

```
<- defiende(javier,finidi).
|   defiende(javier,J1) <- jugador_de(J1,javier).
|   {J1/finidi}
<- jugador_de(finidi,javier).
|   jugador_de(X2,Y2) <- juega_en(X2,E2), entrena(Y2,E2)
|   {X2/finidi, Y2/javier}
<- juega_en(finidi,E2), entrena(javier,E2).
|   juega_en(finidi,betis) <-.
|   {E2/betis}
<- entrena(javier,betis).
|   entrena(javier,betis) <-.
|   {}
<-
```

Resolución SLD

- **Ejemplo de cálculo de respuesta mediante resolución SLD**
  - **Programa**

```
defiende(javier,J)      <- jugador_de(J,javier).
defiende(javier,J)      <- jugador_de(J,javier).
jugador_de(X,Y)         <- juega_en(X,E), entrena(Y,E).
juega_en(finidi,betis) <-.
entrena(javier,betis)  <-.
```

  - **Objetivo**

```
<- defiende(javier,J).
```

  - **Significado del objetivo:** $\neg(\exists J)[defiende(javier, J)]$
  - **Cálculo de respuesta por resolución SLD**

```
<- defiende(javier,J0).
|   defiende(javier,J1) <- jugador_de(J1,javier).
|   {J1/J0}
<- jugador_de(J0,javier)
|   jugador_de(X2,Y2) <- juega_en(X2,E2), entrena(Y2,E2)
|   {X2/J0, Y2/javier}
<- juega_en(J0,E2), entrena(javier,E2).
|   juega_en(finidi,betis) <-.
|   {J=0/finidi, E2/betis}
<- entrena(javier,betis).
|   entrena(javier,betis) <-.
|   {}
<-
```

  - **Respuesta:** {J/finidi}

Resolución SLD

- **Ejemplo de cálculo de respuesta mediante Prolog**
  - **Programa** `ej-1.pl`

```
defiende(javier,J)      :- jugador_de(J,javier).
defiende(javier,J)      :- jugador_de(J,javier).
jugador_de(X,Y)         :- juega_en(X,E), entrena(Y,E).
juega_en(finidi,betis).
entrena(javier,betis).
```

  - **Cálculo de respuesta**

```
?- ['ej-1.pl'].
Yes
?- defiende(javier,J).
J = finidi
Yes
?- trace.
Yes
?- defiende(javier,J).
   Call:  (  7) defiende(javier, _G152) ?
   Call:  (  8) jugador_de(_G152, javier) ?
   Call:  (  9) juega_en(_G152, _L139) ?
   Exit:  (  9) juega_en(finidi, betis) ?
   Call:  (  9) entrena(javier, betis) ?
   Exit:  (  9) entrena(javier, betis) ?
   Exit:  (  8) jugador_de(finidi, javier) ?
   Exit:  (  7) defiende(javier, finidi) ?
J = finidi
Yes
```

Resolución SLD

- **Ejemplo de cálculo de respuesta por resolución SLD en programa recursivo con símbolos de función**
  - **Programa**

```
suma(0,X,X)        <-.
suma(s(X),Y,s(Z)) <- suma(X,Y,Z).
```

  - **Objetivo**

```
<- suma(s(0),s(0),X).
```

  - **Significado del objetivo:** $\neg(\exists \boldsymbol{X})[\boldsymbol{suma}(\boldsymbol{s}(\mathbf{0}), \boldsymbol{s}(\mathbf{0}), \boldsymbol{X})]$
  - **Derivación**

```
<- suma(s(0),s(0),X0).
|   suma(s(X1),Y1,s(Z1)) <- suma(X1,Y1,Z1).
|   {X1/0, Y1/s(0), X0/s(Z1)}
<- suma(0,s(0),Z1).
|   suma(0,X2,X2) <-.
|   {X2/s(0), Z1/s(0)}
<-
```

  - **Respuesta:** {X/s(s(0))}

Resolución SLD

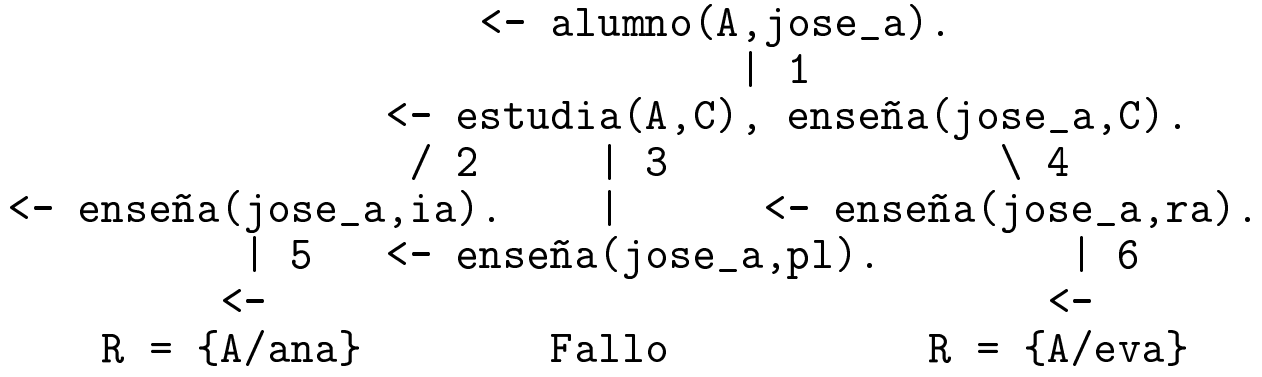
- **Arbol SLD**
  - **Programa**

```
alumno(A,P)       <- estudia(A,C), enseña(P,C).  % 1
estudia(ana,ia)   <-.                            % 2
estudia(ana,pl)   <-.                            % 3
estudia(eva,ra)   <-.                            % 4
enseña(jose_a,ia) <-.                            % 5
enseña(jose_a,ra) <-.                            % 6
enseña(rafael,pl) <-.                            % 7
```

  - **Objetivo**

```
<- alumno(A,jose_a).
```

  - **Arbol SLD**



  - **Ramas de éxito y de fallo**

Resolución SLD

- **Cálculo de respuestas en Prolog**

```
?- alumno(A,jose_a).
A = ana ;
A = eva ;
No
?- trace.
Yes
?- alumno(A,jose_a).
   Call:  (  7) alumno(_G178, jose_a) ?
   Call:  (  8) estudia(_G178, _L130) ?
   Exit:  (  8) estudia(ana, ia) ?
   Call:  (  8) enseña(jose_a, ia) ?
   Exit:  (  8) enseña(jose_a, ia) ?
   Exit:  (  7) alumno(ana, jose_a) ?
A = ana ;
   Redo:  (  8) enseña(jose_a, ia) ?
   Fail:  (  8) enseña(jose_a, ia) ?
   Redo:  (  8) estudia(_G178, _L130) ?
   Exit:  (  8) estudia(ana, pl) ?
   Call:  (  8) enseña(jose_a, pl) ?
   Fail:  (  8) enseña(jose_a, pl) ?
   Redo:  (  8) estudia(_G178, _L130) ?
   Exit:  (  8) estudia(eva, ra) ?
   Call:  (  8) enseña(jose_a, ra) ?
   Exit:  (  8) enseña(jose_a, ra) ?
   Exit:  (  7) alumno(eva, jose_a) ?
A = eva ;
No
```

- **Búsqueda en profundidad**
- **Respuestas múltiples**

Resolución SLD

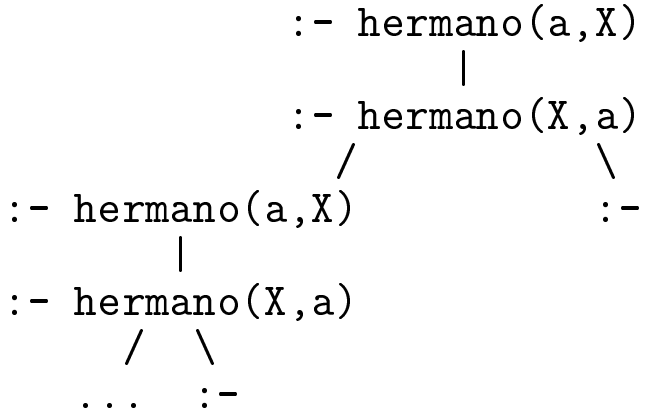
- **Arbol SLD infinito**
  - **Programa**

```
hermano(X,Y) :- hermano(Y,X).
hermano(b,a).
```

  - **Sesión:**

```
?- hermano(a,X).
[WARNING: Out of local stack]
Execution Aborted
```

  - **Arbol:**



  - **Programa**

```
hermano(b,a).
hermano(X,Y) :- hermano(Y,X).
```

  - **Sesión:**

```
?- hermano(a,X).
X = b ;
X = b
Yes
```

Resolución SLD

- **Adecuación y completitud básica de la resolución SLD**
  - Def.: $E(P) = \{A \in BH(P) : P \cup \{\leftarrow A\} \vdash_{SLD} \square\}$
  - Adecuación y completitud:
    $E(P) = \{A \in BH(P) : P \models A\} = \mathrm{MMH}(P)$
- **Adecuación y completitud de la resolución SLD**
  - Sean
    * P un programa lógico y
    * $G = \leftarrow A_1, \ldots, A_n$ un objetivo
  - Adecuación: Si σ es una respuesta computada de $P \cup \{G\}$, entonces $P \models (A_1 \wedge \ldots \wedge A_n)\sigma$
  - Completitud: Si $P \models (A_1 \wedge \ldots \wedge A_n)\theta$, entonces existen σ, γ tales que σ es una respuesta computada de $P \cup \{G\}$ y $\theta = (\sigma\gamma) \restriction \mathrm{Variables}(G)$

Resolución SLD

- **Ejemplo de cálculo de respuesta con variables**
  - **Programa**

```
suma(0,X,X)         <-.
suma(s(X),Y,s(Z)) <- suma(X,Y,Z).
```

  - **Objetivo**

```
<- suma(s(0),X,Y).
```

  - **Significado del objetivo:** $\neg(\exists \boldsymbol{X}, \boldsymbol{Y})[\boldsymbol{suma}(\boldsymbol{s}(\boldsymbol{0}), \boldsymbol{X}, \boldsymbol{Y})]$
  - **Derivación**

```
<- suma(s(0),X0,Y0).
|    suma(s(X1),Y1,s(Z1)) <- suma(X1,Y1,Z1).
|    {X1/0, Y1/X0, Y0/s(Z1)}
<- suma(0,X0,Z1)).
|    suma(0,X2,X2) <-.
|    {X2/X0, Z1/X0}
<-
```

  - **Respuesta:** {Y/s(X)}
  - $\boldsymbol{P} \models \boldsymbol{suma}(\boldsymbol{s}(\boldsymbol{0}), \boldsymbol{X}, \boldsymbol{s}(\boldsymbol{X}))$

Resolución SLD

- **Incompletitud de Prolog**
  - **Programa**

```
hermano(X,Y) :- hermano(Y,X).
hermano(b,a).
```

  - **Sesión:**

```
?- hermano(a,X).
[WARNING: Out of local stack]
Execution Aborted
```

  - $P \models hermano(a, b)$
  - **Causa: Búsqueda en profundidad**
- **Inadecuación de Prolog**
  - **Programa**

```
suma(0,X,X).
suma(s(X),Y,s(Z)) :- suma(X,Y,Z).
igual(0,s(0))     :- suma(s(0),X,X).
```

  - **Sesión**

```
?- igual(X,Y).
X = 0
Y = s(0)
Yes
```

  - $P \not\models igual(0, s(0))$
  - **Causa: Unificación sin test de ocurrencia**

Resolución SLD

- **Multidireccionalidad**
  - **Programa**

```
suma(0,X,X).
suma(s(X),Y,s(Z)) :- suma(X,Y,Z).
```

  - **Resta**

```
?- suma(s(0),X,s(s(s(0)))).
X = s(s(0))
```

  - **Descomposición de `s(s(0))` en dos sumandos**

```
?- suma(X,Y,s(s(0))).
X = 0
Y = s(s(0)) ;
X = s(0)
Y = s(0) ;
X = s(s(0))
Y = 0 ;
No
```

  - **Relación `suma`**

```
?- suma(X,Y,Z).
X = 0
Y = _G149
Z = _G149 ;
X = s(0)
Y = _G149
Z = s(_G149)
Yes
```

Bibliografía

- Flach, P. *Simply Logical (Intelligent Reasoning by Example)* (John Wiley, 1994)
  - Cap. 3: "Logic programming and Prolog"
- Lucas, P. y Gaag, L.v.d. *Principles of Expert Systems* (Addison–Wesley, 1991).
  - Cap. 2 "Logic and resolution"
- Rich, E. y Knight, K. *Inteligencia artificial (segunda edición)* (McGraw–Hill Interamericana, 1994)
  - Cap. 5 "La lógica de predicados"
- Russell, S. y Norvig, P. *Inteligencia artificial (un enfoque moderno)* (Prentice–Hall, 1996)
  - Cap. 9 "La inferencia en la lógica de primer orden"
  - Cap. 10 "Sistemas de razonamiento lógico"
- Winston, P.R. *Inteligencia Artificial (3a. ed.)* (Addison–Wesley, 1994)
  - Cap. 13 "Lógica y prueba de resolución"