

Programación lógica proposicional

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

Secuentes

- Sintaxis de secuentes

- Símbolos proposicionales: $A, A_1, \dots, B, B_1, \dots$

- Secuente: $A_1, \dots, A_n, \leftarrow B_1, \dots, B_m$
con $n \geq 0$ y $m \geq 0$

- Cabeza del secuente: $\{A_1, \dots, A_n\}$

- Cuerpo del secuente: $\{B_1, \dots, B_m\}$

- Procedimientos sobre secuentes

```
;;; (cabeza '(p) <- (q r)) => (P)
(defun cabeza (secuente)
  (first secuente))
```

```
;;; (cuerpo '(p) <- (q r)) => (Q R)
(defun cuerpo (secuente)
  (third secuente))
```

```
;;; (crea-secuente '(p) '(q r)) => ((P) <- (Q R))
(defun crea-secuente (cabeza cuerpo)
  (list cabeza '<- cuerpo))
```

Secuentes

- Transformación de secuentes a fórmulas y cláusulas
 - Secuente: $A_1, \dots, A_n \leftarrow B_1, \dots, B_m$
 - Fórmula: $B_1 \wedge \dots \wedge B_m \rightarrow A_1 \vee \dots \vee A_n$
 - Cláusula $\{A_1, \dots, A_n, \neg B_1, \dots, \neg B_m\}$
- Transformación de secuentes a cláusulas

```
;;; > (secuente->clausula '((a) <- (b c)))
;;; (A (- B) (- C))
;;; > (secuente->clausula '((a) <- ()))
;;; (A)
(defun secuente->clausula (secuente)
  (append (cabeza secuente)
          (mapcar #'complementario
                  (cuerpo secuente))))
```

Cláusulas de Horn

- **Cláusulas de Horn**
 - Un secunte representa una cláusula de Horn si su cabeza tiene como máximo un elemento
 - Cláusulas de Horn son las que no tienen más de un literal positivo
- **Propiedades**
 - Completitud de resolución unidad para cláusulas de Horn
 - Completitud de resolución por entradas para cláusulas de Horn
- **Clasificación de cláusulas de Horn:**
 - Cláusula definida:
 - * Regla: $A \leftarrow B_1, \dots, B_n$ con $n > 0$
 - * Hecho: $A \leftarrow$
 - Objetivo definido:
 - * Objetivo propio: $\leftarrow B_1, \dots, B_n$ con $n > 0$
 - * Objetivo vacío: \leftarrow

Sintaxis de programas lógicos

- Programa lógico

- Def.: Conjunto finito de cláusulas definidas

- Ejemplo de programa lógico

```
es_mamifero <- tiene_pelos
es_mamifero <- da_leche
es_ungulado <- es_mamifero, tiene_pezuñas
es_ungulado <- es_mamifero, rumia
es_jirafa <- es_ungulado, tiene_cuello_largo
es_cebra <- es_ungulado, tiene_rayas_negras
```

```
tiene_pelos <-
tiene_pezuñas <-
tiene_rayas_negras <-
```

- Reglas y hechos del programa

- Representación

```
((es_mamifero) <- (tiene_pelos))
((es_mamifero) <- (da_leche))
((es_ungulado) <- (es_mamifero tiene_pezuñas))
((es_ungulado) <- (es_mamifero rumia))
((es_jirafa) <- (es_ungulado tiene_cuello_largo))
((es_cebra) <- (es_ungulado tiene_rayas_negras))

((tiene_pelos) <- ())
((tiene_pezuñas) <- ())
((tiene_rayas_negras) <- ()))
```

Semántica declarativa

- Base de Herbrand

- Def: La base de Herbrand, $BH(P)$, de un programa lógico, P , es el conjunto de los símbolos proposicionales que aparecen en el programa

- Procedimiento de cálculo

```
;;; > (base-de-Herbrand '((p) <- (q r))
;;;                               ((p) <- (s))
;;;                               ((r) <- ())
;;;                               ((q) <- ())
;;;                               ((p1) <- (s))
;;;                               ((p2) <- (s))))
;;; (P Q R S P1 P2)
(defun base-de-Herbrand (P)
  (simbolos-proposicionales-conjunto-clausulas
   (mapcar #'secuente->clausula P)))
```

- Modelos de Herbrand de un programa lógico

```
;;; > (modelos-de-Herbrand '((p) <- (q r))
;;;                               ((p) <- (s))
;;;                               ((r) <- ())
;;;                               ((q) <- ())
;;;                               ((p1) <- (s))
;;;                               ((p2) <- (s))))
;;; ((P Q R)
;;;  (P Q R P2)
;;;  (P Q R P1)
;;;  (P Q R P1 P2)
;;;  (P Q R S P1 P2))
(defun modelos-de-Herbrand (P)
  (modelos-conjunto-clausulas
   (mapcar #'secuente->clausula P)))
```

Semántica declarativa

- **Propiedades:**

- Sea P un programa lógico. Entonces:

- $BH(P) \models P$

- La intersección de modelos de P es modelo de P

- **Menor modelo de Herbrand**

- Def: M es el menor modelo de Herbrand de P si M es un modelo de P y es subconjunto de todos los modelos de P

- Notación: $MMH(P)$

- $MMH(P) = \bigcap \{M : M \models P\}$

- Procedimiento de cálculo

```
;;; > (menor-modelo-de-Herbrand '((p) <- (q r))
;;;                               ((p) <- (s))
;;;                               ((r) <- ())
;;;                               ((q) <- ())
;;;                               ((p1) <- (s))
;;;                               ((p2) <- (s)))
;;; (P Q R)
(defun menor-modelo-de-Herbrand (P)
  (reduce #'intersection (modelos-de-Herbrand P)))
```

- **Propiedad:**

- $MMH(P) = \{A : P \models A\}$

Semántica de punto fijo

- Operador de consecuencia inmediata

- Definición:

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

- Ejemplo: Programa P :

```
p <- q, r
p <- s
r <-
q <-
p1 <- s
p2 <- s
```

- Consecuencias:

$$\begin{aligned} T_P(\emptyset) &= \{r, q\} \\ T_P(\{r, q\}) &= \{p, r, q\} \\ T_P(\{p, r, q\}) &= \{p, r, q\} \end{aligned}$$

- Encadenamiento adelante

Semántica de punto fijo

- Procedimiento de cálculo

```
;;; > (setf *Prog* '(((p) <- (q r))
;;;      ((p) <- (s))
;;;      ((r) <- ())
;;;      ((q) <- ())
;;;      ((p1) <- (s))
;;;      ((p2) <- (s))))
;;; > (consecuencias-inmediatas *Prog* ())
;;; (R Q)
;;; > (consecuencias-inmediatas *Prog* '(r q))
;;; (P R Q)
;;; > (consecuencias-inmediatas *Prog* '(p r q))
;;; (P R Q)
(defun consecuencias-inmediatas (P I)
  (loop for C in P
        when (subsetp (cuerpo C) I)
        append (cabeza C)))
```

- Menor punto fijo

- Procedimiento de cálculo

```
;;; > (menor-punto-fijo *Prog*)
;;; (P R Q)
(defun menor-punto-fijo (P &optional I)
  (let ((J (consecuencias-inmediatas P I)))
    (if (equal J I) I
        (menor-punto-fijo P J))))
```

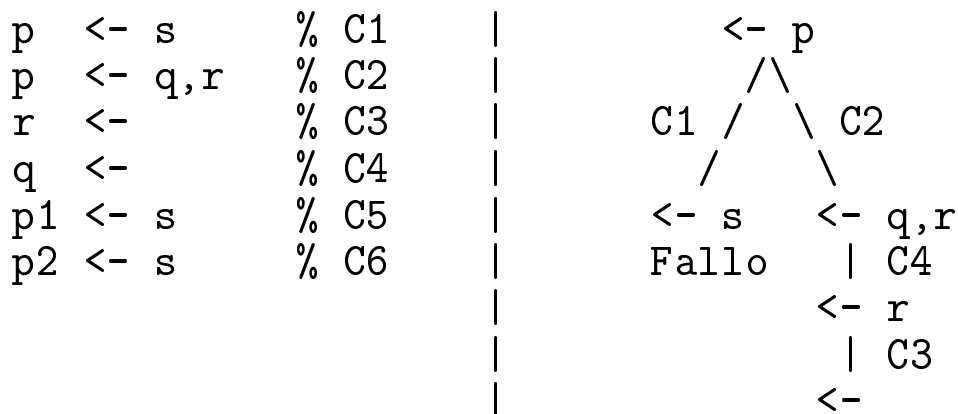
- Notación: $MPF(P)$

- Propiedad: $MPF(P) = MMH(P)$

Resolución SLD

- SLD-resolventes
 - Objetivo: $\leftarrow A_1, \dots, A_{i-1}, A_i, A_{i+1}, \dots, A_n$
 - Cláusula: $A_i \leftarrow B_1, \dots, B_m$
 - SLD-resolvente: $\leftarrow A_1, \dots, A_{i-1}, B_1, \dots, B_m, A_{i+1}, \dots, A_n$
- Doble indeterminismo:
 - Selección del literal del objetivo (regla de computación)
 - Selección de la cláusula del programa (regla de elección)
- Arbol de SLD-resolución

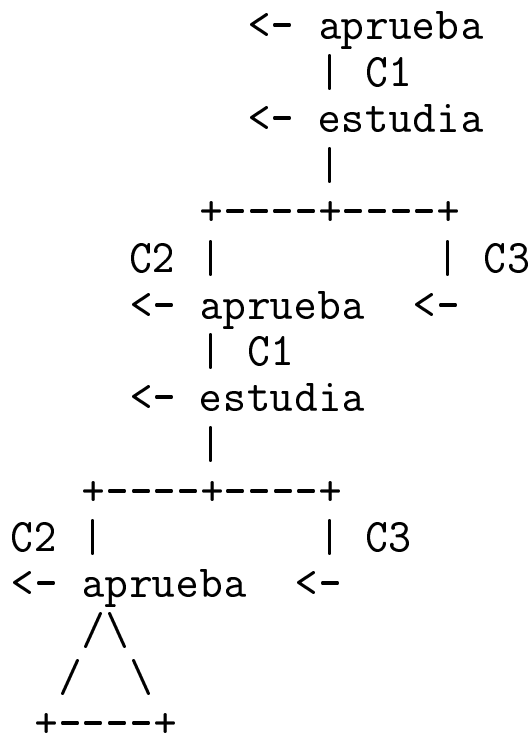
- Ejemplo 1



Resolución SLD

- Ejemplo 2

```
aprueba <- estudia % C1
estudia <- aprueba % C2
estudia <-          % C3
```



- Ramas: de éxito, de fallo, infinita
- Estrategias de búsqueda: profundidad, anchura

Resolución SLD

- SLD-resolución

- Ejemplo:

Programa:

```
p <- s      % C1
p <- q,r    % C2
r <-        % C3
q <-        % C4
p1 <- s     % C5
p2 <- s     % C6
```

SLD-resolución:

```
<- p
  | p <- q,r
  | /
<- q,r
  | q <-
  | /
<- r
  | r <-
  | /
<-
```

Resolución SLD

- Ejemplo 2:

Programa:		SLD-resolución:
a ← b, c		← a
b ← c, d		a ← b, c
c ← e		/
d ←		← b, c
e ←		b ← c, d
		/
		← c, d, c
		c ← e
		← e, d, c
		e ←
		/
		← d, c
		d ←
		/
		← c
		c ← e
		/
		← e
		e ←
		←

Resolución SLD

- SLD–resolución: Resolución Lineal con función de Selección para cláusulas Definidas
- SLD–resolución: resolución lineal por entradas
- Completitud de SLD–resolución
 - P un programa lógico
 - $G = \leftarrow A_1, \dots, A_n$ un objetivo
 - Son equivalentes:
 - * $P \cup \{G\} \vdash_{SLD} \{\}$
 - * $P \cup \{G\}$ es inconsistente
 - * $P \models A_1 \wedge \dots \wedge A_n$
- Prolog proposicional como SLD–resolución
 - Elección de literal en objetivo: El primero
 - Elección de la cláusula: La primera que se equipare
 - Estrategia de búsqueda: En profundidad

Resolución SLD

- Procedimiento de SLD-resolución

- Traza

```
> (setf *programa* '((p) <- (s))
                        ((p) <- (q r))
                        ((r) <- ())
                        ((q) <- ())
                        ((p1) <- (s))
                        ((p2) <- (s))))
> (trace SLD-resolvente)
(SLD-RESOLVENTE)
> (prueba-por-SLD-resolucion
   *programa*
   '(() <- (p)))
(SLD-RESOLVENTE '(NIL <- (P)) '((P) <- (S)))
SLD-RESOLVENTE ==> (NIL <- (S))
(SLD-RESOLVENTE '(NIL <- (P)) '((P) <- (Q R)))
SLD-RESOLVENTE ==> (NIL <- (Q R))
(SLD-RESOLVENTE '(NIL <- (Q R)) '((Q) <- NIL))
SLD-RESOLVENTE ==> (NIL <- (R))
(SLD-RESOLVENTE '(NIL <- (R)) '((R) <- NIL))
SLD-RESOLVENTE ==> (NIL <- NIL)
T
```

Resolución SLD

- Procedimiento

```
(defun prueba-por-SLD-resolucion (P G)
  (if (es-objetivo-vacio G)
      t
      (some #'(lambda (C)
                (and (eql (first (cabeza C))
                        (first (cuerpo G)))
                     (prueba-por-SLD-resolucion
                      P
                      (SLD-resolvente G C))))
        P)))
```

```
(defun es-objetivo-vacio (G)
  (and (null (cabeza G))
        (null (cuerpo G))))
```

```
(defun SLD-resolvente (G C)
  (crea-secuente ()
                  (append (cuerpo C)
                          (rest (cuerpo G)))))
```


Resolución SLD

● Incompletitud de Prolog

```
> (trace SLD-resolvente)
(SLD-RESOLVENTE)
> (prueba-por-SLD-resolucion
  '((aprueba) <- (estudia))
  ((estudia) <- (aprueba))
  ((estudia) <- ()))
  '(() <- (aprueba)))
(SLD-RESOLVENTE ' (NIL <- (APRUEBA))
  '((APRUEBA) <- (ESTUDIA)))
SLD-RESOLVENTE ==> (NIL <- (ESTUDIA))
(SLD-RESOLVENTE ' (NIL <- (ESTUDIA))
  '((ESTUDIA) <- (APRUEBA)))
SLD-RESOLVENTE ==> (NIL <- (APRUEBA))
(SLD-RESOLVENTE ' (NIL <- (APRUEBA))
  '((APRUEBA) <- (ESTUDIA)))

...
> (prueba-por-SLD-resolucion
  '((estudia) <- ()))
  ((aprueba) <- (estudia))
  ((estudia) <- (aprueba)))
  '(() <- (aprueba)))
(SLD-RESOLVENTE ' (NIL <- (APRUEBA))
  '((APRUEBA) <- (ESTUDIA)))
SLD-RESOLVENTE ==> (NIL <- (ESTUDIA))
(SLD-RESOLVENTE ' (NIL <- (ESTUDIA))
  '((ESTUDIA) <- NIL))
SLD-RESOLVENTE ==> (NIL <- NIL)
T
```

Resolución SLD

- Problema de los animales

```
> (trace SLD-resolvente)
(SLD-RESOLVENTE)
> (prueba-por-SLD-resolucion
  '((es_mamifero) <- (tiene_pelos))
  ((es_mamifero) <- (da_leche))
  ((es_ungulado) <- (es_mamifero
                     tiene_pezuñas))
  ((es_ungulado) <- (es_mamifero
                     rumia))
  ((es_jirafa) <- (es_ungulado
                  tiene_cuello_largo))
  ((es_cebra) <- (es_ungulado
                 tiene_rayas_negras))
  ((tiene_pelos) <- ())
  ((tiene_pezuñas) <- ())
  ((tiene_rayas_negras) <- ()))
'((() <- (es_cebra)))
```

Resolución SLD

```
(SLD-RESOLVENTE ' (NIL <- (ES_CEBRA))
                  ' ((ES_CEBRA) <- (ES_UNGULADO
                                TIENE_RAYAS_NEGRAS)))
SLD-RESOLVENTE ==> (NIL <- (ES_UNGULADO
                            TIENE_RAYAS_NEGRAS))
(SLD-RESOLVENTE ' (NIL <- (ES_UNGULADO
                            TIENE_RAYAS_NEGRAS))
                  ' ((ES_UNGULADO) <- (ES_MAMIFERO
                                TIENE_PEUÑAS)))
SLD-RESOLVENTE ==> (NIL <- (ES_MAMIFERO
                            TIENE_PEUÑAS
                            TIENE_RAYAS_NEGRAS))
(SLD-RESOLVENTE ' (NIL <- (ES_MAMIFERO
                            TIENE_PEUÑAS
                            TIENE_RAYAS_NEGRAS))
                  ' ((ES_MAMIFERO) <- (TIENE_PEUÑAS)))
SLD-RESOLVENTE ==> (NIL <- (TIENE_PEUÑAS
                            TIENE_PEUÑAS
                            TIENE_RAYAS_NEGRAS))
(SLD-RESOLVENTE ' (NIL <- (TIENE_PEUÑAS
                            TIENE_PEUÑAS
                            TIENE_RAYAS_NEGRAS))
                  ' ((TIENE_PEUÑAS) <- NIL))
SLD-RESOLVENTE ==> (NIL <- (TIENE_PEUÑAS
                            TIENE_RAYAS_NEGRAS))
(SLD-RESOLVENTE ' (NIL <- (TIENE_PEUÑAS
                            TIENE_RAYAS_NEGRAS))
                  ' ((TIENE_PEUÑAS) <- NIL))
SLD-RESOLVENTE ==> (NIL <- (TIENE_RAYAS_NEGRAS))
(SLD-RESOLVENTE ' (NIL <- (TIENE_RAYAS_NEGRAS))
                  ' ((TIENE_RAYAS_NEGRAS) <- NIL))
SLD-RESOLVENTE ==> (NIL <- NIL)
```

T

Referencias

- Lucas, P. y Gaag, L.v.d. *Principles of Expert Systems* (Addison–Wesley, 1991).
 - Cap. 2 “Logic and resolution”
- Thayse, A. y otros *Aproche logique de l’Intelligence Artificielle. (Vol 1: de la logique classique à la programmation logique)*. (Dunod, 1988)
 - Cap. 1.1 “Calcul des propositions”