Búsqueda e inferencia lógica

Programación Lógica y Prolog

Contenidos

1. Introducción a la Programación Lógica.
2. Programas Definidos: Sintaxis de Edimburgo.
3. Resolución SLD.
4. Intérprete abstracto de un Programa Lógico.
5. Concepto de respuesta.
6. Programación Lógica y Negación.
7. Una implementación práctica: Prolog Estándar.

Anexo: árbol SLD

1. Introducción a la Programación Lógica.

Programación Lógica

- "parte de la informática que se ocupa de la lógica como lenguaje de programación"
  - Programa: conjunto finito de FBF's.
  - Computación: obtención de pruebas formales.

Evolución histórica

- Origen: demostración automática de teoremas + IA
  - Herbrand(30),Davies-Putman(~60), Robinson(65)
- Aparición PL (~70)
  - Kowalsky, Colmerauer, Green, Hayes
- Primer interprete Prolog
  - Colmerauer, Rusell, Marsella 1972
- Primera implementación eficiente
  - Warren, AWM (Máquina Abstracta de Warren) Edimburgo, 1977

Cláusulas definidas

- A lo sumo, un literal positivo:

  $\neg b_1 \vee \neg b_2 \vee \ldots \quad \vee \neg b_n \vee a$, n>=0

- En programación lógica se representa:

  $a \leftarrow b_1, b_2, \ldots \quad ,b_n$

  - donde:
    - $a, b_1, b_2, \ldots \quad ,b_n$ son literales positivos (átomos)
    - todas las variables se consideran cuantificadas universalmente

  - a cabeza de la cláusula
  - $b_1, b_2, \ldots \quad ,b_n$ cuerpo de la cláusula

Caracterización Programas Lógicos

- Programas Definidos
  - Cláusulas Horn o definidas.

- Programas Normales
  - Cláusulas normales: extensión cláusula de Horn, admitiendo literales negativos en cuerpo cláusulas.

- Programas: cualquier FBF.

2. Programas Definidos: Sintaxis de Edimburgo.

Programas Definidos: Sintaxis de Edimburgo

- Términos
  - Constantes
    - Numéricas: *12, -34, 34.87*, …
    - Atómicas: *a, b, ana, estudiante*, …
  - Variables: *x, y, z, x1, x2*, …
  - Funciones: *f(a, x), g(y), madre(ana),…*
- Átomos
  - *p(x), q(a,y), hermano(x,y), madre(ana),…*
- Cláusulas: se construyen con átomos

  padre(x,y) $\leftarrow$ *hombre(x), hijo_de(y,x)*

Programas Definidos: Sintaxis de Edimburgo

- Hecho (cláusula unitaria de programa): $a \leftarrow$
- Regla (cláusula de programa): $a \leftarrow b_1, b_2, \ldots \quad ,b_n$
- Pregunta o meta: $\leftarrow b_1, b_2, \ldots \quad ,b_n$

- Programa: conjunto finito de cláusulas de programa

Ejemplo programa definido

- Hechos
- Reglas
- Pregunta o meta

$\{hombre(juan) \leftarrow,$
$hombre(luis) \leftarrow,$
$hijo\_de(juan, luis) \leftarrow,$
$padre(x,y) \leftarrow hombre(x), hijo\_de(y,x)\}$

$\leftarrow padre(luis, juan)$

3. Resolución SLD

Resolución SLD

Resolución **L**ineal con función de **S**elección para Cláusulas **D**efinidas.

C: $a \leftarrow b_1, b_2, ... b_n$ n>=0, cláusula de programa.

(sin variables comunes con G: renombrar var C)

G: $\leftarrow a_1, a_2, ... a_k$ k>0, pregunta.

f_s: función de selección (regla de cómputo).

$a_s = f_s(G)$, literal seleccionado.

Si a_s y a unifican con umg. θ, se denomina resolvente SLD de G y C a la meta:

$\leftarrow(a_1, a_2, ... a_{s-1}, b_1, b_2, ... b_n, a_{s+1},.... a_k)\ \theta$

Ejemplo resolución SLD

G: $\leftarrow$ *mujer(ana), padre(x, y), hombre(y)*
$f_s(G)$: *padre(x, y)*
C: *padre(u,v)* $\leftarrow$ *hombre(u), hijo_de(v,u)*

Umg de *padre(x,y)* y *padre(u,v)*: $\theta=\{u/x, v/y\}$

Resolvente SLD de *G* y *C*:

$\leftarrow$ *(mujer(ana), hombre(u), hijo_de(v,u), hombre(y))* $\{u/x, v/y\}$

Y aplicando la substitución:

$\leftarrow$ *mujer(ana), hombre(u), hijo_de(v,u), hombre(v)*

En formato estándar

$G: \neg mujer(ana) \vee \neg padre(x, y) \vee \neg hombre(y)$

$C: padre(u,v) \vee \neg hombre(u) \vee \neg hijo\_de(v,u)$

$\theta=\{u/x, v/y\}$

$\neg mujer(ana) \vee \neg padre(x, y) \vee \neg hombre(y)$

$padre(u,v) \vee \neg hombre(u) \vee \neg hijo\_de(v,u)$

$\theta=\{u/x, v/y\}$

$\neg mujer(ana) \vee \neg hombre(v) \vee \neg hombre(u) \vee \neg hijo\_de(v,u)$

Reordenando y transformando:

$\leftarrow mujer(ana), hombre(u), hijo\_de(v,u), hombre(v)$

Derivación SLD (cómputo de G por P)

Sean P un programa y G una meta. Una derivación SLD de P U {G} consiste en tres secuencias, posiblemente infinitas, de:

$G_0=G, G_1, G_2, \ldots$	Metas
$C_1, C_2, C_3, \ldots$	Cláusulas de P renombradas
$\theta_1, \theta_2, \theta_3, \ldots$	Umg's de C_i, G_{i-1}, respectivamente

tal que G_{i+1} es el resolvente SLD de G_i y C_{i+1} usando θ_{i+1}

ESTRATEGIA DE RESOLUCIÓN LINEAL Y POR ENTRADAS

Ejemplo Derivación SLD

G_0: $\leftarrow$ entero(5)

C_1: entero(x_1) $\leftarrow$ entero(x_2), =(x_1,+(x_2,1))

$\theta_1=\{5/x_1\}$

G_1: $\leftarrow$ entero(x_2), =(5,+(x_2,1))

C_2: entero(x_3) $\leftarrow$ entero(x_4), =(x_3,+(x_4,1))

$\theta_2=\{x_2/x_3\}$

G_2: $\leftarrow$ entero(x_4), =(x_2,+(x_4,1)), =(5,+(x_2,1))

C_3: entero(x_5) $\leftarrow$ entero(x_6), =(x_5,+(x_6,1))

$\theta_3=\{x_4/x_5\}$

G_3: $\leftarrow$ entero(x_6), =(x_4,+(x_6,1)), =(x_2,+(x_4,1)), =(5,+(x_2,1))

.
.
.

Refutación SLD

- Def. Derivación SLD de □

Ejemplo Refutación SLD

padre(luis, carlos) $\leftarrow$,
padre(carlos, jorge) $\leftarrow$,
abuelo(x,y) $\leftarrow$ *padre(x,z),padre(z, y)*

G_0: $\leftarrow$ abuelo(luis, jorge) — C_1: abuelo(u,v) $\leftarrow$ padre(u,w),padre(w, v)

θ_1={luis/u, jorge/v}

G_1: $\leftarrow$ padre(luis, w) , padre(w, jorge) — C_2: padre(luis, carlos)

θ_2={carlos/w}

G_2: $\leftarrow$ padre(carlos, jorge) — C_3: padre(carlos, jorge)

$\theta_3=\emptyset$

G_3: $\square$

4. Intérprete abstracto de un Programa Lógico

Intérprete abstracto (no determinista) de un Programa Lógico

InterpreteAbstracto(P, G)
    resolvente ← G;
    mientras (resolvente ≠ Ø) **hacer**
        Q=f_s(resolvente)
        Si (existe cláusula C de P cuya cabeza unifique con Q) **Entonces**
            resolvente ← resolvente_SLD de resolvente y C con umg θ
            G ← G θ
        Sino SalirMientras
        finsi
    finMientras
    Si (resolvente **=** Ø) **Entonces** (**Devolver** G)
    Sino (**Devolver** fallo)

Dos elecciones

- Regla de cómputo: literal sobre el que se resuelve, dado por la función de selección.

- Regla de búsqueda: criterio de selección de la cláusula que resuelve (reduce) la meta.

Efecto regla cómputo, búsqueda

- Regla cómputo: arbitrario.
  - No afecta a la terminación.
  - Quizás orden respuestas.

- Regla de búsqueda: no determinista.
  - Afecta a la terminación.

Regla de búsqueda: afecta a la terminación

C_1: $p(a, b) \leftarrow$
C_2: $p(c, b) \leftarrow$
C_3: $p(x, z) \leftarrow p(x, y), p(y, z)$
C_4: $p(x, y) \leftarrow p(y, x)$

Regla de búsqueda: primera cláusula no utilizada

Regla de cómputo: primer literal a la izquierda

G_0: $\leftarrow p(a, c)$	C_3
G_1: $\leftarrow p(a, y), p(y, c)$	C_1
G_2: $\leftarrow p(b, c)$	C_4
G_3: $\leftarrow p(c, b)$	C_2
G_4: $\square$	

Regla de búsqueda: búsqueda primero en profundidad

Regla de cómputo: primer literal a la izquierda

G_0: $\leftarrow p(a, c)$	C_3
G_1: $\leftarrow p(a, y), p(y, c)$	C_1
G_2: $\leftarrow p(b, c)$	C_3
G_3: $\leftarrow p(b, w), p(w, c)$	C_3
G_4: $\leftarrow p(b, t), p(t, w), p(w, c)$	.
.	.
.	.

Concepto de respuesta

- Def. *P* programa definido, *G* meta definida.

Una respuesta para *P U {G}* es:

  - Una substitución para las variables de *G*
  - "*no*"

Respuesta correcta

- Def. Sean P programa definido,

G meta definida, G: $\leftarrow a_1, a_2, \ldots. a_k$

θ una respuesta de $P \cup \{G\}$

θ es una respuesta correcta para $P \cup \{G\}$ sii

$$P \models \forall(a_1, a_2, \ldots. a_k)\theta$$

($P \models \neg G\theta$ sii $P \cup \{G\theta\}$ inconsistente)

"*no*" es una respuesta correcta para $P \cup \{G\}$ sii

$P \cup \{G\}$ es satisfacible

Respuesta correcta

$P=\{p(x) \leftarrow\}$

$G= \leftarrow p(x)$ $P\cup\{G\}$ respuesta correcta: $\theta=\varnothing$ (*true*)

$G= \leftarrow p(y)$ $P\cup\{G\}$ respuesta correcta: $\theta=\varnothing$ (*true*)

$\theta=\{x/y\}$ es solo una variante alfabética y es equivalente a $\theta=\varnothing$

$G= \leftarrow p(y)$ $P\cup\{G\}$ respuesta correcta: $\theta=\{a/y\}$ (*true*)

$G= \leftarrow p(a)$ $P\cup\{G\}$ única respuesta correcta: $\theta=\varnothing$ (*true*)

$G= \leftarrow p(b)$ $P\cup\{G\}$ única respuesta correcta: $\theta=\varnothing$ (*true*)

$G= \leftarrow p(f(y))$ $P\cup\{G\}$ respuesta correcta: $\theta=\varnothing$ (*true*)

Respuesta correcta

$P=\{p(x) \leftarrow\}$

$G= \leftarrow p(f(x))$

$P\cup\{G\}$ respuesta correcta: $\theta=\varnothing$ (*true*)

PERO

$P=\{p(x, x) \leftarrow\}$

$G= \leftarrow p(x, f(x))$

$P\cup\{G\}$ única respuesta correcta: *no*

Respuesta computada

- Def. Sean *P* programa definido,

G meta definida / *P U {G}* tiene una refutación SLD,
$\theta_1, \theta_2, \theta_3, \ldots \quad \ldots \theta_n$ la secuencia de umg’s utilizada en la refutación SLD

θ es una respuesta computada para *P U {G}* sii θ es la substitución obtenida seleccionando de $\theta_1\ \theta_2\ \theta_3 \ldots \quad \ldots \theta_n$ las ligaduras de las variables que ocurren en *G*.

Respuesta computada

padre(luis, carlos) $\leftarrow$,
padre(carlos, jorge) $\leftarrow$,
abuelo(x,y) $\leftarrow$ *padre(x,z),padre(z, y)*

G_0: $\leftarrow$ abuelo(x,y)

C_1: abuelo(u,v) $\leftarrow$ padre(u,w),padre(w, v)

θ_1={x/u, y/v}

G_1: $\leftarrow$ padre(x,w) , padre(w, y)

C_2: padre(luis, carlos)

θ_2={luis/x, carlos/w}

G_2: $\leftarrow$ padre(carlos, y)

C_3: padre(carlos, jorge)

θ_3={jorge/y}

G_3: $\square$

Respuesta computada

$\theta_1 \theta_2 \theta_3 = \{x/u, y/v\} \{luis/x, carlos/w\} \{jorge/y\}$

$\theta_1 \theta_2 \theta_3 = \{x/u, y/v\} \{luis/x, carlos/w, jorge/y\}$

$\theta_1 \theta_2 \theta_3 = \{luis/u, jorge/v, luis/x, carlos/w, jorge/y\}$

$G = \leftarrow abuelo(x,y)$

Variables que ocurren en la meta original: x, y

Respuesta computada de $P \cup \{G\}$: $\theta = \{luis/x, jorge/y\}$

Teorema solidez resolución SLD

Sea *P* un programa definido y *G* una meta definida.

Toda respuesta computada de *P U {G}* es una respuesta correcta de *P U {G}*

Teorema complitud resolución SLD

Sean P un programa definido, G una meta definida y θ una respuesta correcta de $P \cup \{G\}$

$\exists$ Respuesta computada σ y substitución γ tales que θ y $\sigma\gamma$ tienen el mismo efecto sobre las variables de G

(La respuesta computada puede ser más general que la correcta)

Diferencias respuesta correcta/computada (I)

$P=\{q(x) \leftarrow\}$

$G= \leftarrow q(y)$

Única respuesta computada: $\theta=\varnothing$

Respuestas correctas: $\theta_1=\{a/y\}$, $\theta_2=\{b/y\}$,... ya que $\leftarrow q(a)$, $\leftarrow q(b)$ etc., son consecuencias lógicas de P

La respuesta computada es más general.

Diferencias respuesta correcta/computada (II)

$P=\{q(x) \leftarrow\}$

$G= \leftarrow q(y)$

Única respuesta computada: $\sigma=\varnothing$

Una respuesta correcta: $\theta=\{a/y\}$

$\exists\sigma=\varnothing$ y $\exists\gamma=\{a/y\}$ / $\sigma\gamma=\theta$

Particularizando la respuesta computada, más general, se obtiene cualquier respuesta correcta.

Teorema independencia de la regla de cómputo

- Sean *P* programa definido, *G* meta definida / *P U {G}* tiene una refutación SLD con respuesta computada θ.

Para cualquier otra regla de cómputo, *R*, existe una refutación SLD de *P U {G}* vía *R* con respuesta computada θ' / $G\theta'$ es una variante alfabética de $G\theta$.

6. Programación Lógica y Negación.

Programación lógica y negación

- Programa definido: conjunto de hechos y reglas que describen explícitamente *qué es cierto*, sin información explicita sobre *qué es falso*

- Dado Programa *P*, meta *G*, definidos, sólo podemos obtener respuestas computadas, σ, que también son correctas: sólo podemos derivar consecuencias lógicas

Programación Lógica: solo podemos derivar consecuencias lógicas

P={estudiante(juan) $\leftarrow$, profesor(luis) $\leftarrow$}

G= $\leftarrow$ estudiante(luis)

- Única respuesta (correcta y computada): "no"

  Porque P $\not\models$ estudiante(luis)

- Incluso para la meta normal G= $\leftarrow \neg$estudiante(luis), la respuesta, en el ámbito de programas definidos y resolución SLD debería ser "no"

  Porque P $\not\models \neg$ estudiante(luis)

Suposición de mundo cerrado (SMC)

- Regla de inferencia:

  Sean P programa definido y a átomo básico.

  Si $P \not\models a$, inferir $\neg a$

- Observaciones:
  - SMC natural y efectiva en contexto de bases de datos.
  - Regla de inferencia no-monotónica.
  - Problemática en el contexto de Programación Lógica, pues no se puede garantizar el cómputo de $P \models a$.

Suposición de mundo cerrado (SMC)

- Regla de inferencia:

  Sean P programa definido y a átomo básico.

  Si $P \not\models a$, inferir $\neg a$

P={estudiante(juan) $\leftarrow$, profesor(luis) $\leftarrow$}

G= $\leftarrow \neg$estudiante(luis)

Respuesta (computada): $\theta = \emptyset$

Necesidad negación

- Teóricamente, innecesaria: “Toda función computable en el sentido de Turing se puede computar con un programa definido” (1977, Tärlund)

- En la práctica, su ausencia limita capacidad expresiva
  - ¿Cómo definir que dos conjuntos son distintos sin la negación?

Negación por fallo

- Regla de inferencia "Negación por fallo":

  Sean P programa definido y a átomo básico.

  Si $P \not\models a$ tiene una prueba finita, inferir $\neg a$

- Prueba finita de $P \not\models a$ (informal):
  - Número finito de derivaciones SLD.
  - Todas finitas.
  - Ninguna permite derivar a.

Prueba finita de $P \not\models a$

P={estudiante_grado(juan) ←, estudiante_doctorado(luis) ← ,
estudiante(x) ← estudiante_grado(x),
estudiante(x) ← estudiante_doctorado(x),
profesor(x) ← estudiante_doctorado(x)}

G= ← profesor(juan)

← profesor(juan)

profesor(y) ← estudiante_doctorado(y)

← estudiante_doctorado(juan)

- No hay ramas infinitas
- Todas las ramas, finitas, son ramas fallo: no permiten derivar □

Negación por fallo

P={estudiante_grado(juan) $\leftarrow$, estudiante_doctorado (luis) $\leftarrow$,
estudiante(x) $\leftarrow$ estudiante_grado(x),
estudiante(x) $\leftarrow$ estudiante_doctorado(x),
profesor(x) $\leftarrow$ estudiante_doctorado(x)}

G= $\leftarrow \neg$profesor(juan)

Respuesta: $\theta = \varnothing$

Porque existe una prueba finita de P $\not\models$ profesor(juan)

Negación por fallo

P={estudiante_grado(juan) $\leftarrow$, estudiante_doctorado (luis) $\leftarrow$,
estudiante(x) $\leftarrow$ estudiante_grado(x),
estudiante(x) $\leftarrow$ estudiante_doctorado(x),
profesor(x) $\leftarrow$ estudiante_doctorado(x)}

G= $\leftarrow \neg$profesor(luis)

¿Respuesta?

Negación por fallo

$P = \{p(x, z) \leftarrow q(x, y), p(y, z),$

$\quad p(x, x) \leftarrow ,$

$\quad q(a, b) \leftarrow \}$

$G = \leftarrow \neg p(a, b)$

¿Respuesta?

No es una prueba finita de *P* $\not\models$ *a*

P: C_1: p(x, z) ← q(x, y), p(y, z)
C_2: p(x, x) ←
C_3: q(a, b) ←

G= ← p(a, b)

Regla de cómputo: 1er literal a la izquierda.

← p(a, b)

C_1 {a/x, b/z}

← q(a, y), p(y, b)

{b/y} C_3

← p(b, b)

C_1 C_2

← q(b, u), p(u, b) $\square$

Finito, pero existe una rama éxito que termina con $\square$,
luego *P* $\models$ *p(a, b)*

Negación por fallo

$P = \{p(x, z) \leftarrow q(x, y), p(y, z),$

$\quad p(x, x) \leftarrow ,$

$\quad q(a, b) \leftarrow \}$

$G = \leftarrow \neg p(a, b)$

Respuesta: “no”

Porque no existe una prueba finita de $P \not\models p(a, b)$

Prueba finita de $P \nvDash a$

P: C_1: $p(x, z) \leftarrow q(x, y), p(y, z)$
C_2: $p(x, x) \leftarrow$
C_3: $q(a, b) \leftarrow$

G= $\leftarrow p(c, b)$

Regla de cómputo: 1er literal a la izquierda.

$\leftarrow$ p(c, b)

{c/x, b/z} **C_1**

$\leftarrow$ q(c, y), p(y, b)

- No hay ramas infinitas
- Todas las ramas, finitas, son ramas fallo: no permiten derivar □

Negación por fallo

$P = \{p(x, z) \leftarrow q(x, y), p(y, z),$

$\quad p(x, x) \leftarrow ,$

$\quad q(a, b) \leftarrow \}$

$G = \leftarrow \neg p(c, b)$

¿Respuesta?

SMC, Negación por Fallo: regla de inferencia no monotónica

$P=\{estudiante(juan) \leftarrow, profesor(luis) \leftarrow\}$

$G= \leftarrow \neg estudiante(luis)$

Respuesta (computada): $\theta=\varnothing$

- Si añadimos a P la cláusula estudiante(luis) $\leftarrow$

$P'=\{estudiante(juan) \leftarrow, profesor(luis) \leftarrow, estudiante(luis) \leftarrow\}$

$G= \leftarrow \neg estudiante(luis)$

Respuesta (computada): “no”

Negación por fallo y programas definidos: Resolvente SLDNF

- Sean P programa definido, G_i meta normal, $l_s=f_s(G_i)$ literal seleccionado

El resolvente SLDNF de P y G_i sobre l_s, G_{i+1}, es:

a) l_s literal positivo:
   - resolvente SLD de G_i y C_{i+1}, con C_{i+1} cláusula de programa cuya cabeza unifique con l_s

b) l_s literal negativo básico y existe prueba finita $P \not\models \neg l_s$
   - meta resultante de eliminar l_s de G_i

Negación por fallo y programas definidos: Resolvente SLDNF

P={estudiante_grado(juan) ←, estudiante_grado(luis) ← , estudiante(x) ← estudiante_grado(x), estudiante(x) ← estudiante_doctorado(x), profesor(x) ← estudiante_doctorado(x)}

G= ← estudiante(juan), ¬profesor(juan)

← estudiante(juan), ¬profesor(juan)

{juan/y} estudiante(y) ← estudiante_grado(y)

← estudiante_grado(juan), ¬profesor(juan)

estudiante_grado(juan) ←

← ¬profesor(juan)

¬profesor(juan) átomo negativo, básico

existe prueba finita **P⊭profesor(juan)**

□

Programas normales

- Cláusulas Normales: admiten átomos negativos en el cuerpo de las cláusulas.
- Negación: negación por fallo.
- Regla de inferencia SLDNF:
  - Similar a SLDNF con programas definidos y metas normales.
  - Técnicamente, más compleja.
  - Admite literales negativos con variables.

Principales resultados en programas normales

- No se mantiene el teorema de independencia de la regla de cómputo.
- SLDNF no es sólida.

Ejemplo de programa normal

$P=\{animal(snoopy) \leftarrow ,$
$animal(lamia) \leftarrow ,$
$serpiente(lamia) \leftarrow ,$
$gusta(elena, x) \leftarrow animal(x), \neg serpiente(x)\}$

Programas normales: no se mantiene el teorema de independencia de la regla de cómputo

A) Regla de cómputo: 1er literal a la izquierda.

G= ← gusta(elena, x)

{y/x} gusta(elena, y) ← animal(y), ¬serpiente(y)

← animal(y), ¬serpiente(y)

{snoopy/y} animal(snoopy) ←

← ¬serpiente(snoopy)

□

Programas normales: no se mantiene el teorema de independencia de la regla de cómputo

B) Regla de cómputo: 1er literal a la derecha.

G= $\leftarrow$ gusta(elena, x)

$\{y/x\}$ gusta(elena, y) $\leftarrow$ animal(y), $\neg$serpiente(y)

$\leftarrow$ animal(y), $\underline{\neg\text{serpiente}(y)}$

No existe una prueba finita de P$\not\models$ serpiente(y) θ

Respuesta computada: “no”

SLDNF no es sólida en programas normales

- Ver ejemplo anterior.
- SLDNF puede derivar una respuesta computada (“no”) que no es respuesta correcta.

7. Una implementación práctica: Prolog Estándar

Prolog éstandar

- Implementación secuencial modelo de programación lógica.
- Programas normales.
- Regla de cómputo: primer literal a la izquierda.
- Regla de búsqueda: primero en profundidad .
  - (implementación: backtracking)

Dificultades: complitud

- Prolog no es completo (por la regla de búsqueda)
  - Incluso en programas definidos puede no encontrar una refutación cuando esta existe.

Prolog no es completo

- Por la regla de búsqueda
  - Incluso en programas definidos puede no encontrar una refutación cuando esta existe.

P={entero(X):- entero(Y), X is Y+1.

entero(0).}

?- entero(Z)

Dificultades: solidez

- No es sólido, pues SLDNF no lo es.
- Sugerencia: evitar variables libres en literales negativos seleccionados.
  - No incluyéndolos.
  - Intentando forzar ligadura operacionalmente.

Dificultades: solidez

- No es sólido, pues SLDNF no lo es

P={animal(snoopy).
animal(lamia).
serpiente(lamia).
gusta(elena, x) :- ¬serpiente(x), animal(x).}

Mejor

P={animal(snoopy).
animal(lamia).
serpiente(lamia).
gusta(elena, x) :- animal(x), $\neg$serpiente(x).

Desviaciones modelo lógico

- Ausencia chequeo de ocurrencias
- Respuestas computadas no correctas
  - *P={p(X, f(X)). q(a):-p(X,X).}, G=?-q(a)*.
- Bucles infinitos
  - P={q(a):-p(X,X), p(X, f(X)):-p(X,X).}, G=?-q(a).

Desviaciones modelo lógico

- Corte: ! (cut)
- Árbol SLD (programa *P* definido, meta *G* definida): árbol de derivaciones SLD para la meta *G* con el programa *P*
- Efecto Corte: impide explorar algún subárbol
- Afecta complitud programa definidos y normales
- Afecta solidez programas normales(Prolog)

Anexo: árbol SLD

Árbol SLD

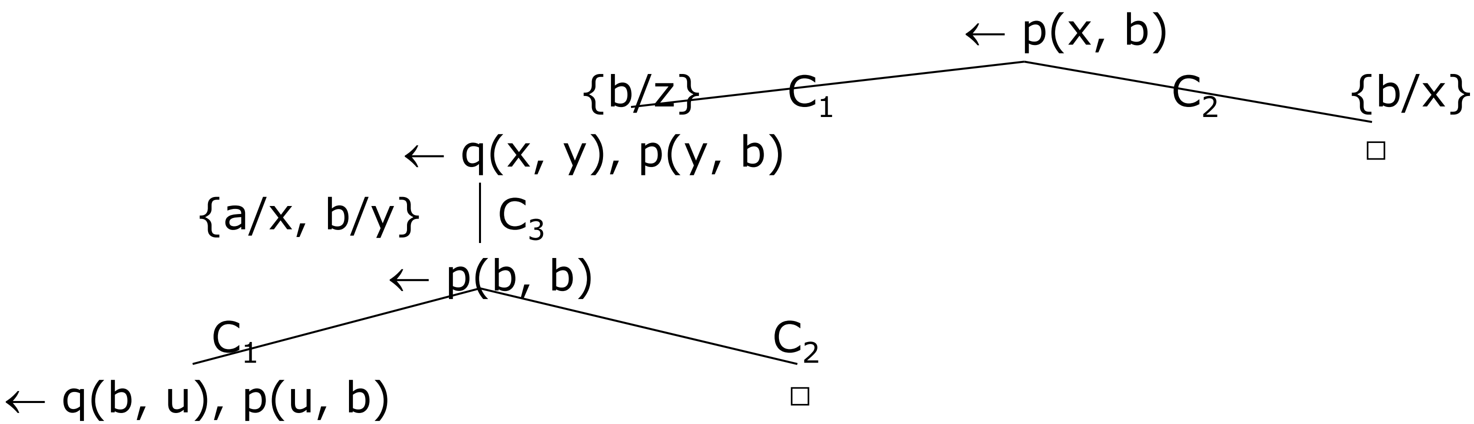
- Sean P un programa definido y G una meta definida.
- Un árbol SLD para P U {G} es un árbol que cumple:
  a) El nodo raíz es G
  b) Cada nodo del árbol es una meta definida (posiblemente vacía)
  c) Dado un nodo cualquiera con meta G_i y $a_s = f_s(G_i)$ el literal seleccionado por la función de selección, el nodo tiene un hijo por cada cláusula de programa cuya cabeza unifique con a_s
  d) Los nodos con cláusulas vacías no tienen hijos.

Ejemplo Árbol SLD (I)

P: C_1: $p(x, z) \leftarrow q(x, y), p(y, z)$
C_2: $p(x, x) \leftarrow$
C_3: $q(a, b) \leftarrow$

G= $\leftarrow p(x, b)$

Regla de cómputo: 1er literal a la izquierda.

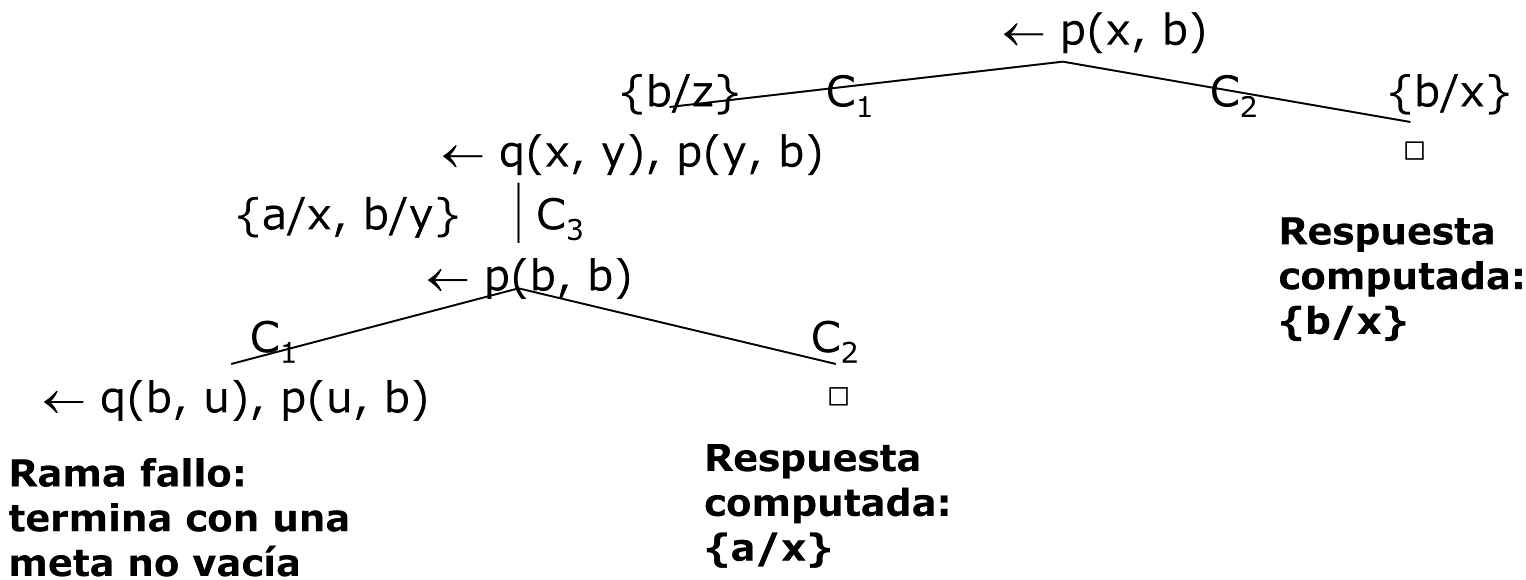


Ejemplo Árbol SLD (I) y respuestas computadas

P: C_1: $p(x, z) \leftarrow q(x, y), p(y, z)$
C_2: $p(x, x) \leftarrow$
C_3: $q(a, b) \leftarrow$

G= $\leftarrow p(x, b)$

Regla de cómputo: 1er literal a la izquierda.

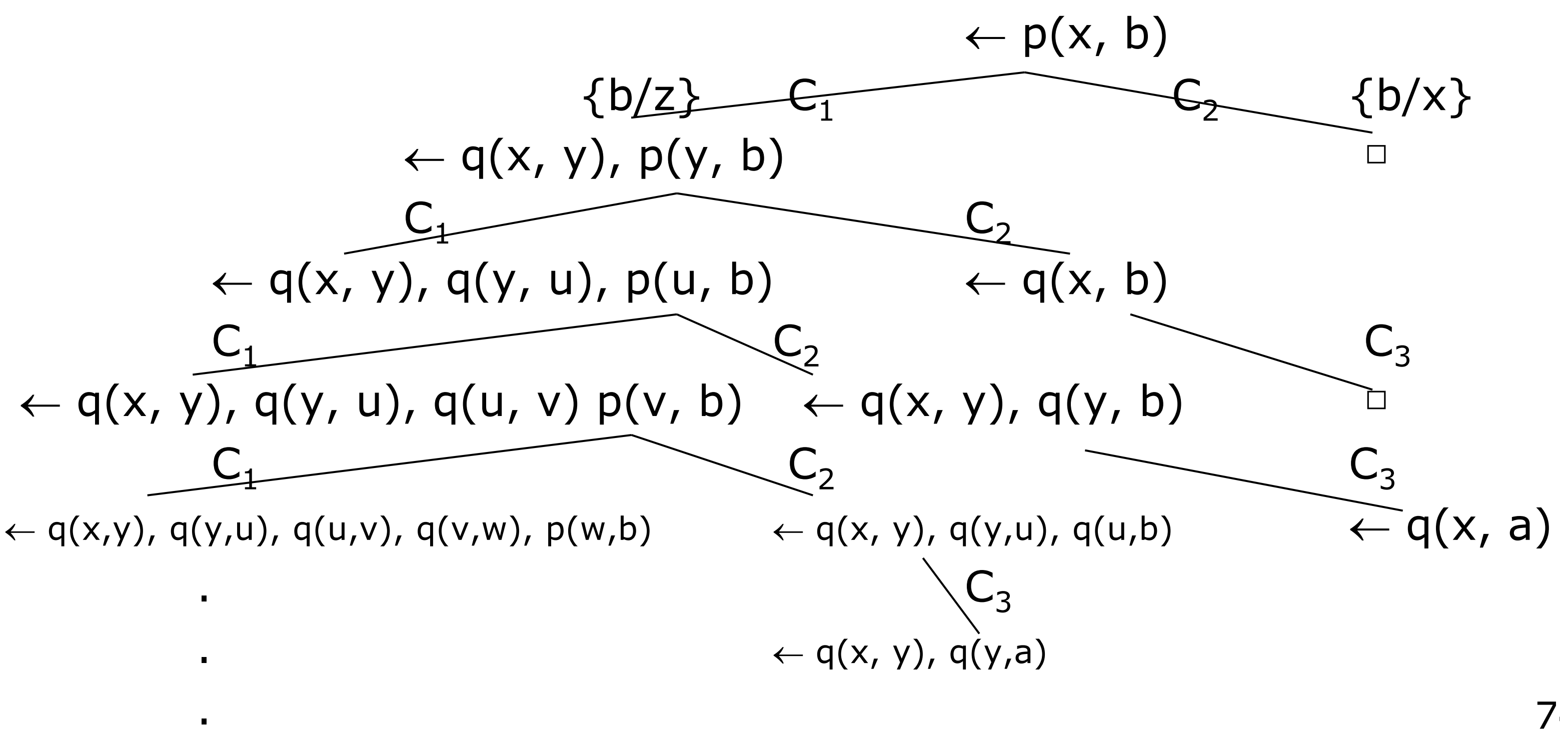


Ejemplo Árbol SLD (II)

P: C_1: p(x, z) ← q(x, y), p(y, z)
C_2: p(x, x) ←
C_3: q(a, b) ←

G= ← p(x, b)

Regla de cómputo: 1er literal a la derecha.



Ejemplo Árbol SLD (II) y respuestas computadas

P: C_1: $p(x, z) \leftarrow q(x, y), p(y, z)$
C_2: $p(x, x) \leftarrow$
C_3: $q(a, b) \leftarrow$

G= $\leftarrow p(x, b)$

Regla de cómputo: 1er literal a la derecha.