Conferencia 3: Programación Declarativa

M.Sc Dafne García de Armas

Curso 2018-2019

Tema: Principio de Resolución SLD. Árbol de derivación SLD.

En la conferencia anterior estudiamos los distintos componentes sintácticos que caracterizan un lenguaje de programación lógica. Por tanto, se conoce que la programación lógica hace uso de un tipo especial de lógica, la denominada Lógica de Cláusulas Definidas (LCD) para la cual la estrategia de resolución SLD es una regla de inferencia completa que puede emplearse como principio operacional semántico de los programas lógicos.

En esta conferencia desarrollaremos la *semántica procedimental* del intérprete de un programa Prolog, es decir el procedimiento mediante el cual dicho intérprete evalúa un objetivo con respecto a un programa. El procedimiento se basa en una regla de inferencia denominada *Principio de Resolución*, la que es utilizada en asociación con el proceso de unificación ya estudiado para computar los valores de las variables en un objetivo.

Las reglas de inferencia deductiva constituyen el método fundamental de generación y justificación lógica de pasos en una demostración y son las que caracterizan a un razonamiento como deductivo. La estructura general de una regla de inferencia deductiva es la siguiente:

Premisas

Conclusión

donde ***Premisas*** denota un conjunto de pasos previos, por lo mismo ya justificados, que ocurren en una demostración y ***Conclusión*** denota el nuevo paso que se inserta en la demostración al aplicar la regla.

Todo proceso de deducción consta al menos implícitamente de una regla de inferencia deductiva denominada *regla de separación o modus ponens* cuyo esquema general es el siguiente:

Modus Ponens (MP)
$\boldsymbol{T} \vdash \mathbf{A}$
$\underline{\boldsymbol{T} \vdash \mathbf{A} \Rightarrow \mathbf{B}}$
$\boldsymbol{T} \vdash \mathbf{B}$

donde T denota una teoría y **A**, **B** son fórmulas, cuya lectura puede ser la siguiente: "A partir de que en una demostración exista un paso de la forma $\boldsymbol{T} \vdash \mathbf{A}$ y otro paso de la forma $T \vdash \mathbf{A} \Rightarrow \mathbf{B}$, entonces se puede insertar $T \vdash \mathbf{B}$ como un nuevo paso en la demostración. Note que $T \vdash \mathbf{A}$ y $T \vdash \mathbf{A} \Rightarrow \mathbf{B}$ son las premisas de la regla y $T \vdash \mathbf{B}$ su conclusión.

En el área de la Ciencia de la Computación que estudia la automatización de la demostración y el razonamiento en general fue definida por Robinson [1966] una regla de inferencia denominada Principio de Resolución (PR), la cual se aplica a fórmulas en forma clausal. Su versión más simple para la lógica proposicional es la siguiente (siendo A un literal, B y C cláusulas):

Principio de Resolución (PR)
$\boldsymbol{T} \vdash \mathbf{A} \vee \mathbf{B}$
$\underline{\boldsymbol{T} \vdash \neg \mathbf{A} \vee \mathbf{C}}$
$\boldsymbol{T} \vdash \mathbf{B} \vee \mathbf{C}$

Observe los siguientes resultados en la aplicación de PR en la lógica proposicional:

$$\boldsymbol{T} \vdash \mathbf{p} \vee \mathbf{q} \vee \mathbf{r}$$
$$\underline{\boldsymbol{T} \vdash \neg \mathbf{p}}$$
$$\boldsymbol{T} \vdash \mathbf{q} \vee \mathbf{r}$$

Vea que en el siguiente ejemplo PR hace la función de Modus Ponens:

$$\boldsymbol{T} \vdash \mathbf{p}$$
$$\underline{\boldsymbol{T} \vdash \neg \mathbf{p} \vee \mathbf{q}}$$
$$\boldsymbol{T} \vdash \mathbf{q}$$

Verifique que en este otro ejemplo PR hace la función de otra regla de inferencia de la lógica proposicional:

$$\boldsymbol{T} \vdash \neg \mathbf{p} \vee \mathbf{q}$$
$$\underline{\boldsymbol{T} \vdash \neg \mathbf{q} \vee \mathbf{r}}$$
$$\boldsymbol{T} \vdash \neg \mathbf{p} \vee \mathbf{r}$$

Como se verá la aplicación del PR en las cláusulas de LPO descansa en el proceso de unificación: antes de aplicar PR entre los literales de primer orden L y $\neg L'$ es necesario determinar si existe un umg σ tal que $L\sigma \cong L'\sigma$.

Considere el programa familia constituido por las siguientes cláusulas:

(P1) padre (luis,alicia).
(P2) padre (luis,josé).
(P3) padre (jose,ana).
(M1) madre (alicia,dario).
(A1) abuelo(X,Y) :- padre(X,Z),madre(Z,Y).
(A2) abuelo(X,Y) :- padre(X,Z),padre(Z,Y).

Observe que las cláusulas (P1)-(P3) definen *extensionalmente* la relación padre/2, es decir, se afirma entre qué pares de individuos se mantiene la relación en este programa (lo mismo puede decirse de la relación madre/2). Por otra parte, (A1)-(A2) definen *intencionalmente* la relación abuelo/2: no se listan los hechos que afirman entre qué pares de individuos se mantiene la relación, sino que se define la relación abuelo/2 en términos de la relación padre/2 y madre/2 que están definidas extensionalmente en el programa.

Debemos detenernos una vez más sobre este aspecto esencialmente definitorio de los programas lógicos. Como lenguaje de programación basado en la lógica, Prolog es eminentemente *declarativo*, como se ha dicho, es uno de los exponentes fundamentales de la *programación declarativa*. Toda fórmula de LPO es un enunciado o proposición declarativa. Las cláusulas de un programa *definen* predicados o relaciones, es decir, responde al *Qué?* Pero para computar soluciones, es decir, para evaluar un objetivo tal como **:- abuelo(X,darío)** (¿Quién es el abuelo de Darío?) a partir del programa Familia el intérprete de Prolog se comporta como una caja negra a la pregunta *Cómo evaluar?* Revelemos esta caja negra.

Principio de Resolución-SLD (PR-SLD)

Desarrollaremos la versión específica del principio de resolución que utiliza Prolog, denominada Resolución-SLD. Dado un programa definido P y un objetivo definido G, para probar la inconsistencia de $P \cup \{G\}$, se parte del objetivo G que se toma como cláusula inicial y se resuelve esta con alguna cláusula del programa P. En cada paso, la resolución genera un nuevo objetivo G_i el cual se resuelve igualmente utilizando solo cláusulas del programa P. Este procedimiento continúa hasta encontrar la cláusula vacía, fallar o entrar en una derivación infinita.

Antes de enunciar formalmente el principio de resolución SLD definimos un concepto clave para su aplicación.

Definición: Una ***regla de selección R*** es una función que selecciona de cualquier conjunto finito de átomos un átomo denominado el *átomo seleccionado*. Formalmente decimos que: La regla de selección R es una función que, cuando se aplica a un objetivo G, selecciona uno y solo uno de los átomos de G.

En la siguiente definición se debe recordar que la cabeza de una cláusula es siempre un literal positivo, mientras que cualquier sub-objetivo de un objetivo es un literal negativo. Por lo tanto si ambos literales unifican, entonces se puede aplicar el Principio de Resolución a ambas cláusulas de acuerdo con la siguiente:

Definición: (*Principio de Resolución-SLD*)

Sea G_i el objetivo $:- A_1, \ldots, A_m, \ldots, A_k$.
C la cláusula $A :- B_1, \ldots, B_q$
R una regla de selección,
entonces un nuevo objetivo G_{i+1} *se deriva* de G_i y de *C* usando el umg σ mediante la regla de selección *R*, si se cumplen las siguientes condiciones:

i. A_m es el átomo de G_i seleccionado por la regla de selección *R*,
ii. $A_m \sigma = A\sigma$ (siendo σ un umg de A_m y *A*),
iii. G_{i+1} es el objetivo: $:- (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)\sigma$ (G_{i+1} es un *resolvente* de G_i y de *C*).

G_i y *C* se denominan las *premisas o cláusulas padres* y G_{i+1} la *conclusión* o el *resolvente* de la derivación.

Ejemplo de aplicación de la Resolución-SLD:

$G_0 :- p(a), \neg q(b)$.
$C\ p(X) :- r(X, a)$.

Se obtiene el resolvente
$G_1 :- r(a, a), \neg q(b)$.

a partir de *C* seleccionando mediante *R* el sub-objetivo a la extrema izquierda de *G0* y usando el umg $\sigma_1 = \{a/X\}$.

Las siglas "**SLD**" se refieren a las siguientes características de la especialización del principio de resolución en la programación lógica:

S: señala la utilización de una ***regla de selección*** *R* del sub-objetivo en el objetivo como premisa. La regla elimina el indeterminismo en la selección del sub-objetivo.

L: la resolución es ***lineal***, en el sentido de que siempre se toma como cláusula padre para la próxima aplicación de resolución el último resolvente obtenido, en nuestro caso, siempre el último objetivo generado.

D: apunta a la utilización sólo de ***cláusulas definidas*** en la aplicación de resolución.

Al seleccionar una cláusula resulta necesario renombrar todas las variables que ocurren en la misma con nuevas variables que no han ocurrido en el desarrollo de la derivación, con lo cual se evita principalmente un injustificable fracaso del proceso de unificación por chequeo de ocurrencia.

Definición: Una *variante de cláusula C de un programa P* es una cláusula *C'* a la cual se le ha aplicado una sustitución σ que renombra todas sus variables. σ se denomina sustitución renombrante.

Ejemplo:
Si *C* es la cláusula $p(X) :- q(Y)$ y *C'* es la cláusula $p(Z) :- q(W)$, entonces *C'* es una variante de la cláusula *C*, pues se obtiene de la siguiente forma: *C'=C*σ donde σ = *{Z/X, W/Y}* es una sustitución *renombrante*.

Para evitar que cada lector cree su propio método de renombrar variables, se explicará cuál será el método que se debe seguir para así lograr uniformidad en los futuros procesos de interpretación de un objetivo:

Dada una cláusula C de un programa Prolog, la cual ha sido identificada con un nombre N, cuando se necesite por primera vez una variante de dicha cláusula, se llamará N1, es decir agregar al nombre N el número 1 y a toda variable que aparezca en C se le adiciona este nuevo identificador. Posteriormente, cada vez que se necesite una i-ésima variante de C (i>1) la nueva variante se denominará Ni y a toda variable que aparezca en C se le adiciona este nuevo identificador.

Ejemplos:
Dada la cláusula identificada por A2 del programa familia antes expuesto:

(A2) abuelo(X,Y) :- padre(X,Z),padre(Z,Y).

La primera variante de cláusula de A2 es:

(A21) abuelo(XA21,YA21) :- padre(XA21,ZA21),padre(ZA21,YA21).

La segunda variante de cláusula de A2 es:

(A22) abuelo(XA22,YA22) :- padre(XA22,ZA22),padre(ZA22,YA22).

Definición Derivación SLD:
Sea *P* un programa, sea *G* un objetivo y sea *R* una regla de selección. Una *derivación-SLD* a partir de $P \cup \{G\}$ mediante *R* es una sucesión finita o infinita $G = G_0, G_1, \ldots$ de objetivos, una sucesión $C_1, C_2, \ldots$ de variantes de cláusulas en *P* que no comparten variables y una sucesión $\sigma_1, \sigma_2, \ldots$ de umg's, tales que cada G_{i+1} se deriva por resolución-SLD de G_i y de C_{i+1} usando σ_{i+1} mediante *R*.

$$G_0 \underset{\Longrightarrow}{[C_1,\theta_1]} G_1 \underset{\Longrightarrow}{[C_2,\theta_2]} G_2 \underset{\Longrightarrow}{[C_3,\theta_3]} \ldots \underset{\Longrightarrow}{[C_{n-1},\theta_{n-1}]} G_{n-1} \underset{\Longrightarrow}{[C_n,\theta_n]} G_n$$

Dado programa *P* y un objetivo *G*, existen dos grupos de derivaciones-SLD de $P \cup \{G\}$: las derivaciones finitas o infinitas. Las derivaciones finitas pueden ser de ‘éxito denominadas refutación SLD o de fallo denominadas derivación fracaso.

Tipos de derivaciones-SLD

Definición: Una ***refutación-SLD*** de $P \cup \{G\}$ mediante R es una derivación-SLD finita cuyo último objetivo es la cláusula vacía □.

Ejemplo 1: Refutación-SLD.

Considere el Programa Familia dado por el conjunto de cláusulas siguientes y el objetivo :- abuelo(luis, darío).

P1:padre(luis,alicia).
P2:padre(luis,jose).
P3:padre(jose,ana).
M1:madre(alicia,dario).
A1:abuelo(X,Y):-padre(X,Z),madre(Z,Y).
A2:abuelo(X,Y):-padre(X,Z),padre(Z,Y).

La siguiente sucesión de cláusulas constituye una refutación de dicho conjunto. En la derivación se utiliza una regla de selección *R* que siempre selecciona el sub-objetivo a la extrema izquierda del objetivo.

G0: :-abuelo(luis,dario).
A11: abuelo(XA11,YA11):-padre(XA11,ZA11),madre(ZA11,YA11).
umg1 = {luis/XA11,dario/YA11}

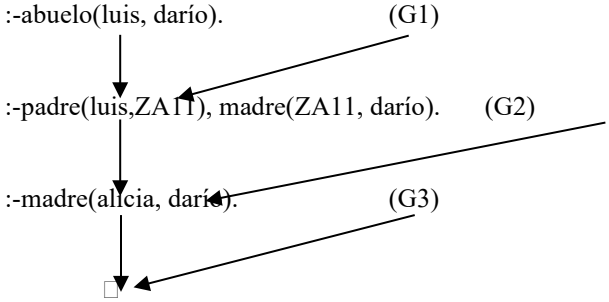
G1: :-padre(luis,ZA11),madre(ZA11,dario).
P11: padre(luis,alicia).
umg2 = {alicia/ZA11}

G2: :-madre(alicia,dario).
M11: madre(alicia,dario).
umg3 = {} = ε

G3: □ (el resolvente de G2 y M11 es la cláusula vacía)
Como no hay variables que instanciar en el objetivo el intérprete se limita a dar como respuesta un ¨si¨.

σ3 = ε : cuando no hay variables, el unificador es la *sustitución idéntica*.

La deducción en forma de árbol muestra la característica lineal de la aplicación de la resolución-SLD:



Lo que computa un programa lógico: *Sustitución (respuesta) computada* de una derivación-SLD

Considere el conjunto de cláusulas Familia ∪ {:- abuelo(luis, X.)}. Nuestro objetivo con este programa no se satisface al demostrar que ∃(X)abuelo(luis, X) es una consecuencia lógica de familia. En verdad nuestro objetivo es una solicitud (*query*) al programa para que nos proporcione como valor de X el nombre de algún nieto de Luis. Como podemos verificar en la siguiente derivación, la unificación computa este valor para X:

Ejemplo 2: computar el valor X del objetivo :-abuelo(luis, X).

(G0) :- abuelo(luis, X).
A11: abuelo(XA11, YA11):- padre(XA11,ZA11),madre(ZA11,YA11).
σ1 = *{luis/XA11, X/YA11}* – recordar que la sustitución *X/YA11* se lee: "YA11 será sustituida por X"

(G2) :- padre(luis,ZA11),madre(ZA11,X). –resolvente de (G0) y (A11)
P11: padre(luis, alicia).
σ2 = *{alicia/ZA11}*

(G3) :-madre(alicia, X).
M11:madre(alicia, darío).
σ3 = *{darío/X}*

(G4) □

Hallando la composición de los tres umg generados en la derivación se tiene que
σ1 σ2 σ3 = { luis/XA11, darío /YA11, alicia/ZA11, darío/X }

denominaremos a tal sustitución la ***sustitución computada de la derivación-SLD.***

Luego, además de suministrar la deducción de *abuelo(luis, X)*, la derivación-SLD a través del proceso de unificación que se desencadena **ofrece un valor para las variables que ocurran en el objetivo** *G0*, mediante su sustitución computada, en el caso que nos ocupa, *X = darío*.

Denominaremos *sustitución respuesta computada (src)* a la restricción de la sustitución computada de la refutación-SLD de *G0* a las variables que ocurren en el objetivo *G0*. Formalmente,

Definición: Sea *P* un programa, *G* un objetivo y *R* una regla de selección. Dada una refutación-SLD de *P* ∪ *{G}* mediante *R*, una *src* de *P* ∪ *{G}* es la sustitución σ que se obtiene al restringir la composición de la sucesión de los umg's $\sigma_1, \sigma_2, ..., \sigma_n$ generados en la refutación a las variables que aparecen en G. En el ejemplo sería *src = {darío/X}*

Derivación-SLD fracasada o fracaso

Definición: Una derivación-SLD *fracasada* o *fracaso* de P ∪ *{G}* mediante R es una derivación-SLD finita $G = G_0, G_1, ... G_m$ tal que $G_m \neq \square$ y no existe $C \in P$ tal que pueda obtenerse un resolvente G_{m+1} a partir de G_m y C.

Ejemplo 3: Derivación-SLD fracaso:

Considere el conjunto de cláusulas Familia ∪ {:- abuelo(luis, alicia)}. La siguiente sucesión de cláusulas constituye una derivación-SLD fracasada de dicho conjunto. En la derivación se utiliza una regla de selección R que siempre selecciona el sub-objetivo a la extrema izquierda del objetivo. Introduciremos, como es costumbre en la programación lógica, el átomo fail como último paso de un fracaso.

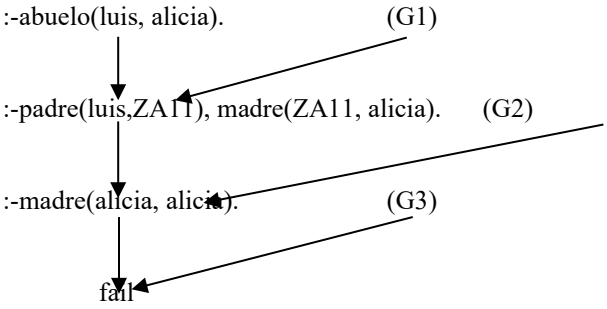
(G0) :- abuelo(luis, alicia).
A11: abuelo(XA11, YA11):- padre(XA11,ZA11),madre(ZA11,YA11).
σ1 = *{luis/XA11, alicia/YA11}*

(G1) :- padre(luis,ZA11),madre(ZA11,alicia).
P11: padre(luis, alicia).
σ2 = *{alicia/ZA11}*

(G2) :-madre(alicia, alicia).

(G3) fail –no existe una variante de cláusula que unifique con (G2) para continuar la derivación

La deducción representada en forma de árbol muestra la característica lineal de la aplicación de resolución:



Derivación-SLD infinita

Si una derivación-SLD no es una refutación-SLD o un fracaso, entonces es una derivación-SLD *infinita*. Más adelante veremos casos de derivaciones infinitas.

Ejemplo: Dado el programa P1: p(f(X)):-p(X). y el objetivo :-p(X).
La siguiente derivación SLD es infinita, pues cualquier objetivo única con una variante de la regla P1.

G0: :-p(X).
P11: p(f(XP11)):-p(XP11).
umg1 = {f(XP11)/X}

G1: :-p(XP11).
P12: p(f(XP12)):-p(XP12).
umg2 = {f(XP12)/XP11}

G2: :-p(XP12).
P13: p(f(XP13)):-p(XP13).
umg3 = {f(XP13)/XP12}

.

.

Corrección y completitud de la resolución-SLD

Corrección:
Una regla de inferencia deductiva debe ser, *correcta (sana)*, es decir, *toda interpretación que sea un modelo de sus premisas debe ser un modelo de sus conclusiones*. Luego si una regla de inferencia es correcta su conclusión es una consecuencia lógica de sus premisas, preservando o extendiendo la verdad de sus premisas a la conclusión.

Completitud:
Una regla de inferencia es *completa si es posible obtener todas las consecuencias lógicas de una teoría de primer orden con la regla*.

Queda de estudio independiente al lector demostrar ambas cosas para la resolución SLD.

Arboles de derivación SLD

Un modelo o paradigma de programación debe dar respuestas a tres dimensiones fundamentales necesarias para computar soluciones a problemas, es decir, para diseñar y programar algoritmos que den solución a clases de problemas. Estas dimensiones son: *representación, proceso y control*.

Los procedimientos que se emplean para solucionar problemas y clases de problemas se denominan de manera general *algoritmos*.

Todo algoritmo da respuesta e interrelaciona las tres dimensiones necesarias para hallar la solución de un problema cumpliendo con la fórmula:

Algoritmo = representación+proceso+control

Representación: El *conocimiento* necesario para resolver una clase de problemas así como los datos del problema específico de la clase que se quiere resolver debe tener una representación adecuada en un lenguaje. Como se ha visto el lenguaje de la lógica de cláusulas definidas es la solución que se ofrece al problema de la **representación en Prolog**.

Proceso: Un algoritmo debe incluir operaciones que actuando sobre los datos de un problema produzcan las transformaciones necesarias hasta lograr como salida una respuesta correcta al problema. Como se ha estudiado un intérprete que transforma un objetivo (entrada) y suministra valores de sus variables (salida) de acuerdo con un programa basado en la **regla de inferencia Principio de Resolución-SLD** y la unificación es la solución fundamental de Prolog a la dimensión proceso.

Control: El proceso (operaciones) que aplica un algoritmo debe ser dirigido al hallazgo de una solución de manera eficiente. En el caso de Prolog estamos hablando del **Árbol de derivación SLD.**

Espacio de búsqueda de la programación lógica:

Muchos problemas de difícil solución algorítmica pueden resolverse tratándolos como un problema de búsqueda en un espacio de estados que denominaremos espacio de búsqueda (EB). Un EB consta de un estado inicial, estados intermedios y estados finales que señalan que se ha alcanzado el éxito en la resolución del problema. El espacio de búsqueda puede representarse mediante una estructura arborescente o un grafo.

Un procedimiento de prueba por refutación SLD puede entenderse como un proceso de búsqueda en un espacio de estados constituidos por los propios objetivos que se generan en cada paso de la resolución. En el caso concreto de un programa definido P y un objetivo G, el conjunto de todas las posibles derivaciones SLD para $P \cup G$ se puede representar como un árbol. A continuación formalizamos el concepto de árbol de búsqueda o árbol de derivación SLD dando una definición constructiva que permite generar este tipo de estructuras.

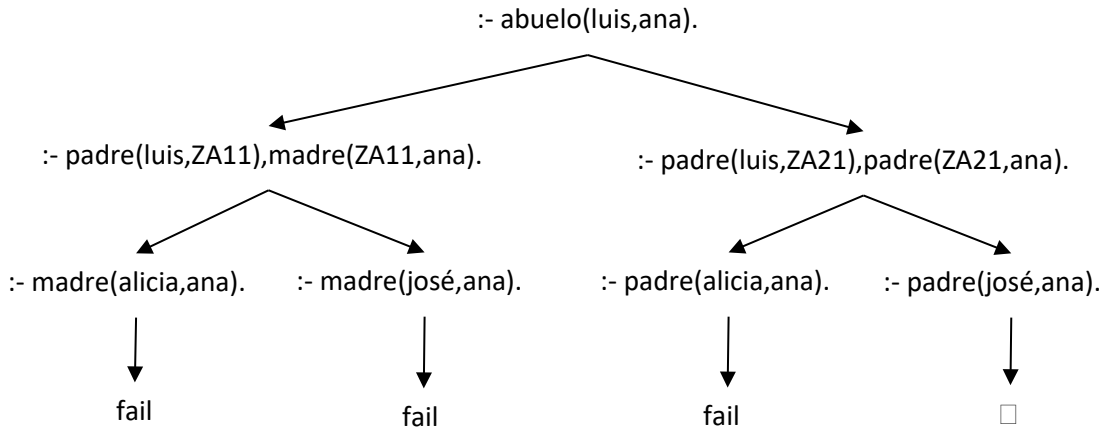
Espacio de búsqueda de la programación lógica: árbol de derivación-SLD

Dado $P \cup \{G\}$, lo primero es tener una representación en potencia del espacio de búsqueda del objetivo *G*. La siguiente definición ofrece una solución a este problema.

Definición: Dado $P \cup \{G\}$, el *árbol de derivación-SLD de G* es un árbol etiquetado que satisface las siguientes condiciones:
-El nodo raíz tiene como etiqueta a *G*, que denominaremos G_0.
- Si el árbol contiene un nodo etiquetado por G_i y existe una variante de cláusula C_{i+1} de $C \in P$ tal que G_{i+1} es un resolvente de G_i y C_{i+1} mediante la regla de selección *R*, entonces existe un nodo hijo de G_i etiquetado por G_{i+1}. El arco que conecta a ambos nodos tiene como etiqueta a C_{i+1}.

El árbol de derivación-SLD de G puede contener ramas *triunfos*, correspondientes a refutaciones-SLD de G, ramas *fracasos*, correspondientes a derivaciones-SLD fracasadas y ramas *infinitas*, correspondientes a derivaciones-SLD infinitas.

Ejemplo:
Árbol de derivación-SLD para Familia ∪ {:- abuelo(luis, ana) }



Se puede observar que un objetivo inicial o cualquier resolvente que pase a ser el nuevo objetivo en la derivación-SLD puede estar constituido por varios sub-objetivos. Un ejemplo de ello en el árbol dado:

:- padre(luis, ZA11), madre(ZA11, ana).

Se puede adoptar, como se establece en la definición formal dada (véase la Conferencia anterior), una regla de selección fija del sub-objetivo a expandir. En el árbol dado la regla adoptada es expandir el primer sub-objetivo o el sub-objetivo a la **extrema izquierda.**

Para cualquier programa P y objetivo G, está demostrado que, independiente de cuál sea la regla de selección R adoptada, si $P \cup \{G\}$ es refutable, entonces el correspondiente árbol de derivación-SLD de *G* tiene una rama triunfo.

Luego, dada una regla de selección *R* fija, cada rama del árbol de derivación-SLD de *G* corresponde a una derivación-SLD de *G* a partir de *P*, existiendo una correspondencia biunívoca entre el conjunto de las derivaciones-SLD de *G* y las ramas del árbol de derivación-SLD de *G*.

Por lo anterior, dado $P \cup \{G\}$, el árbol de derivación-SLD de G constituye una representación estructurada del espacio de búsqueda de *G*.

Nótese que cada nodo del árbol es una cláusula objetivo obtenido como resolvente SLD del nodo padre y una cláusula del programa. Los nodos que no pueden resolverse con ninguna cláusula del programa se dice que son las hojas del árbol. Los nodos hojas son o bien la cláusula vacía □ o nodos de fallo (denotados por el átomo fail). Debido a la forma en la que se construye, cada rama de un árbol de derivación SLD es una derivación SLD para $P \cup \{G\}$, esto es, dada una regla de selección R fija, existe una correspondencia biunívoca entre el conjunto de las derivaciones SLD para $P \cup \{G\}$ y las ramas del árbol de derivación SLD para $P \cup \{G\}$. Un _árbol de derivación puede contener ramas triunfos, correspondientes a refutaciones SLD, ramas fracasos correspondientes a derivaciones SLD de fallo y ramas infinitas, correspondientes a derivaciones SLD infinitas.

Ejercicio: construya el árbol de derivación-SLD para Concatena ∪ {**:-** concatena(X, [1, 2], Y).}

Cada regla de selección R da lugar a un _árbol de derivación SLD para $P \cup \{G\}$ distinto. Esto es fácil de comprobar, por ejemplo, obtenga el _árbol de derivación para Familia [{:-abuelo (luis, ana).} utilizando como regla de selección aquella que \selecciona el _átomo más a la derecha del objetivo".

Como podrá observar, la regla de selección influye en la forma y el tamaño de los árboles de derivación SLD. Sin embargo, cualquiera que sea la regla de selección empleada, si $P \cup \{G\}$ es inconsistente entonces el árbol de derivación SLD contendrá una rama triunfo. Esta propiedad se denomina independencia de la regla de selección y puede formularse de manera alternativa para expresar que el conjunto de sustituciones respuestas computadas (src) obtenido por árboles de derivación SLD construidos con diferentes reglas de selección para un mismo programa P y objetivo G es siempre el mismo. Este resultado justifica que la selección de un literal en un objetivo se puede realizar arbitrariamente sin que afecte el resultado de la computación.

Teorema (Independencia de la Regla de Selección) Sea P un programa definido y G un objetivo definido. Si existe una refutación SLD para $P \cup \{G\}$ con sustitución respuesta computada θ, usando una regla de selección R, entonces existirá también una refutación SLD para $P \cup \{G\}$ con sustitución respuesta computada θ', usando cualquier otra regla de selección R', tal que θ' es una variante de θ.

Contrariamente a la regla de selección, la elección de la cláusula del programa que se utiliza en cada paso de resolución es determinante a la hora de alcanzar el éxito en una derivación. Se dice que la selección de las cláusulas del programa debe hacerse de forma indeterminista, pues no hay modo de saber cuál es la cláusula adecuada que, si se aplica en el paso de resolución actual, garantizará un progreso hacia la obtención de la cláusula vacía. Por tanto, si se quiere mantener la completitud del procedimiento de prueba por refutación SLD es necesario probar a resolver con todas las clausulas y generar toda la combinatoria posible.

La definición de árbol de derivación SLD tiene en cuenta este indeterminismo en la selección de las cláusulas del programa de modo que, al generar los hijos de cada nodo, se intenta la resolución del nodo padre con todas las cláusulas del programa. De esta forma, el árbol de derivación SLD para un programa P y un objetivo G define todo el espacio de búsqueda generado por la estrategia de resolución (con una regla de selección dada) para $P \cup \{G\}$.

Puede comprobarse fácilmente con un ejemplo, que para una misma regla de selección, distintas reglas de ordenación de las cláusulas del programa producen arboles SLD diferentes en los que sus ramas aparecen permutadas.

Definición (Regla de ordenación de cláusulas) Una regla de ordenación de cláusulas es un criterio por el que se fija el orden en que las cláusulas del programa se intentan resolver con un nodo del árbol de derivación SLD, para generar sus correspondientes nodos hijos.

Además de la regla de selección y el orden en que se eligen las cláusulas, que como se ha señalado tienen impacto en la forma y tamaño del _árbol de derivación SLD, para buscar las ramas de éxito en el árbol importante la estrategia con la que este se recorre.

Definición (Regla de búsqueda) Una regla de búsqueda es una estrategia para recorrer un árbol SLD en busca de una rama de éxito.

Estrategias de búsqueda

Una vez fijadas una regla de selección y ordenación de las cláusula concretas, es evidente que el árbol de derivación SLD podría generarse completamente de forma explícita. Sin embargo, en la práctica esto no se realiza así. En lugar de generar explícitamente el árbol de derivación SLD y luego buscar en él la cláusula vacía dada una regla de búsqueda, lo que se hace es proporcionar una representación implícita y solamente se generan explícitamente aquellas partes por las que avanza la estrategia de búsqueda. Existen dos estrategias de búsqueda fundamentales las cuales analizaremos a continuación.

Búsqueda primero en amplitud o anchura (breadth-first search)
La estrategia búsqueda primero en amplitud (BPA) recorre el árbol por niveles, visitando el nodo menos profundo y a la izquierda de entre aquellos todavía no visitados. Esto es, se expanden todos los nodos a profundidad d antes que los nodos de profundidad d+1. La implementación se realiza de la siguiente manera: en cada iteración del procedimiento de refutación se construye el nivel siguiente del árbol de derivación SLD, generando todos los nodos hijos de los estados objetivo que componen el nivel actual. Si en el siguiente nivel se encuentra la cláusula vacía, se termina con éxito; si no, se repite el proceso.

Búsqueda primero en profundidad (depth-first search)
La estrategia búsqueda primero en profundidad (BPP) recorre el árbol visitando primero el nodo más profundo y a la izquierda de entre aquellos todavía no visitados, es decir, se expanden primero los nodos no expandidos más profundos. Este tipo de estrategia suele implementarse bien mediante una técnica de vuelta atrás (backtracking).

Dado que en BPA se exploran todas las posibilidades de resolución de los estados objetivos de un nivel con las cláusulas del programa, esta estrategia dado un objetivo G y un programa P genera el árbol de derivación SLD para $P \cup \{G\}$ y mantiene la completitud del procedimiento de refutación SLD. Sin embargo, si bien esta estrategia es completa y _optima de acuerdo a nuestro criterio de optimalidad (generar una rama triunfo de profundidad mínima.), resulta ser muy costosa en cuanto a complejidad temporal y espacial, ya que el crecimiento del espacio de búsqueda es exponencial con respecto al nivel de profundidad del árbol de derivación SLD.

La estrategia BPP, no es óptima ni completa (analizar ejemplo siguiente), sin embargo, al mantener solo información sobre los estados de la rama que está inspeccionando en el momento actual, su complejidad espacial es mucho menor representando una solución aceptable al problema de la memoria.

Ejemplo: Dado el programa definido:

C1: p(X,Z):-q(X,Y),p(Y,Z).
C2: p(X,X).
C3: q(a,b).

La estrategia BPP no puede encontrar la cláusula vacía para el objetivo definido **:-p(X,b).** cuando se elije como regla de selección "seleccionar el literal más a la derecha del objetivo considerado" y como regla de ordenación "seleccionar las cláusulas de arriba hacia abajo", ya que la primera rama del árbol de derivación SLD es una rama infinita y se pierde en ella.

Un procedimiento de prueba por refutación SLD (el control en un programa lógico) queda entonces completamente especificado fijando: (i) una regla de selección; (ii) una regla de ordenación; y (iii) una regla de búsqueda. Las dos primeras fijan la forma y tamaño del árbol de derivación SLD y la última la manera de recorrerlo.

El intérprete de Prolog

En esta sección describimos el procedimiento de prueba por refutación SLD estándar de los sistemas Prolog tradicionales.

Regla de selección
La regla de selección de Prolog trata los objetivos como secuencias de átomos y siempre selecciona para resolver el átomo situado más a la izquierda dentro del objetivo considerado.

Regla de ordenación de la cláusulas
Prolog intenta unificar el átomo seleccionado por la regla de selección con las cabezas de las cláusulas del programa considerándolas en el orden textual en que aparecen dentro del programa. Esto supone que la regla de ordenación toma las cláusulas de arriba hacia abajo. Contrariamente a lo que sucede en los procedimientos de prueba de la lógica de predicados, las cláusulas no se consideran como un conjunto sino como una secuencia.

Regla de búsqueda
Prolog utiliza como regla de búsqueda la estrategia de búsqueda primero en profundidad con retroceso cronológico (chronological backtrack) (RC), lo cual denotaremos por BPP-RC. La utilización de RC implica que dado un fracaso en la aplicación de resolución SLD a un objetivo en un paso, regresamos al objetivo padre.

Dado un programa definido P y un objetivo definido G el control para la búsqueda de ramas triunfos (ramas que terminan en la cláusula vacía) en Prolog es el siguiente:

1. Dado un objetivo actual Gi, no se generan (expanden) simultáneamente todos sus resolventes (nodos hijos) utilizando todas las cláusulas del programa P que unifiquen con Gi, sino se aplica solo una (la seleccionada por la regla de ordenación), digamos Cj, generando de esta forma un único resolvente Gij, que será el nuevo objetivo actual.

Si no es posible expandir el nuevo objetivo (el resolvente Gij es un nodo fracaso), entonces se retrocede a su objetivo padre Gi para aplicarle una nueva cláusula del programa $C_k \in P$, con $k \neq j$, es decir, una clausula no empleada aún con la que pueda obtenerse un nuevo resolvente (hijo).

2. Si al resolver el objetivo actual Gi no existen cláusulas que unifiquen con él o se han probado con todas las cláusulas del programa y todos sus resolventes conducen a nodos fracaso, entonces hay un fracaso en la resolución del objetivo actual y se retrocede a su objetivo padre G_{i-1}.

3. Si la resolución del objetivo raíz (objetivo inicial G) fracasa, entonces el proceso de búsqueda termina con el fracaso de la demostración del objetivo G.

4. Si al resolver el objetivo actual Gi se genera como nodo hijo la cláusula vacía, entonces el proceso de búsqueda termina exitosamente la demostración del objetivo inicial G. En este caso, se ha recorrido una rama triunfo del árbol de derivación SLD correspondiente a una refutación SLD.

El proceso de búsqueda además de terminar con éxito o con fracaso, puede en ocasiones transcurrir indefinidamente sin encontrar una refutación habiéndola como se puede apreciar con el siguiente programa y el objetivo :-p(a,c).

Ejemplo:

P1: p(a,b).
P2: p(c,b).
P3: p(X,Z):-p(X,Y),p(Y,Z).
P4: p(X,Y):-p(Y,X).

Analice el ejemplo anterior y verifique que cualquiera sea la regla de ordenación de las cláusulas que se adopte, la regla de selección que se elija (o el orden de objetivos en el cuerpo de las cláusulas que se fije), BPP-RC genera una rama infinita que impide alcanzar la rama triunfo.

Este ejemplo muestra que la estrategia de búsqueda BPP-RC rompe con la completitud de la resolución SLD, pues en ocasiones la búsqueda se profundiza por una rama infinita impidiendo refutar el objetivo.

Otro aspecto a destacar sobre la utilización de la estrategia de búsqueda BPP-RC es que esta requiere de la atención, por parte del programador, del orden en que se escriben las cláusulas en un programa y, por tanto, del orden en que serán tomadas por el intérprete en cada paso de resolución, así como del orden en que se disponen los sub-objetivos en el cuerpo de una cláusula. Como se puede comprobar en el siguiente ejemplo, intercambiar el orden de las cláusulas que definen un predicado puede conllevar a que el proceso de búsqueda no termine.

Ejemplo: Considere el programa append/3 definido con el siguiente orden de las cláusulas:

append ([X|Y],Z,[X|W]):-append(Y,Z,W).
append ([],X,X).

Como se puede comprobar, para el objetivo ":-concatena(X,[1,2],Y).", la estrategia BPP-RC no termina, mientras que si lo hace con éxito si se invierte el orden de las cláusulas.

Intercambiar el orden en que se disponen los sub-objetivos en el cuerpo de una cláusula resulta ser también decisivo en un programa Prolog para alcanzar un resultado.