Clase 3.1: El intérprete de Prolog. Principio de Resolución-SLD y estrategia de búsqueda BPP-RC.

Profesora: Dra. Yisel Garí

En esta clase 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)**
. **T |- A**
. **T |- A ⇒B**
. ———————
. **T |- B**

donde **T** denota una teoría y **A**, **B** son fórmulas y cuya lectura puede ser la siguiente: A partir de que en una demostración exista un paso de la forma **T |- A** y otro paso de la forma **T |- A ⇒B** , entonces se puede insertar **T |- B** como un nuevo paso en la demostración. Note que **T |- A** y **T |- A ⇒B** son las premisas de la regla y **T |- 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)**
. **T |- A ∨ B**
. **T |- ¬A ∨ C**
. ———————
. **T |- B ∨ C**

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 que 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 que 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. Previamente definiremos un concepto que juega un papel fundamental en la aplicación de la Resolución-SLD.

Definición: Una *regla de selección* R es una función que selecciona de cualquier conjunto finito de átomos un átomo denominado el *átomo seleccionado*.
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:

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

i) A_m es el átomo seleccionado por la regla de selección R,
ii) $A_m\sigma_{i+1} = A\sigma_{i+1}$ (siendo σ_{i+1} un umg de A_m y A),
iii) G_{i+1} es el objetivo : $-(A_1, ..., A_{m-1}, B_1, ..., B_q, A_{m+1}, ..., A_k)\sigma_{i+1}$
G_{i+1} es un resolvente de G_i y C_{i+1}. G_i y C_{i+1} se denominan las premisas o cláusulas padres y G_{i+1} la conclusión o resolvente de la derivación.

Ejemplo de aplicación de la Resolución-SLD:

G_0 : `:- p(a), q(b).`
C_1 : `p(X) :- r(X,a).`
G_1 : `:- r(a,a), q(b).` (a partir de C_1 seleccionando mediante R el sub-objetivo a la extrema izquierda de G_0 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.

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: $C' = C\sigma$ donde $\sigma = Z/X, W/Y$ 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, dicha variante 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: 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 $G = G_0, G_1, ...$

de objetivos, una sucesión C_0, C_1, ... de variantes de cláusulas en P que no comparten variables y una sucesión $\sigma_0, \sigma_1, ...$ de umg's, tales que cada $G_i + 1$ se deriva por resolución-SLD de G_i y de $C_i + 1$ usando $\sigma_i + 1$ mediante R.

Está claro que dado un programa P y un objetivo G, existe un número infinito de derivaciones-SLD de $P \cup G$. Por ejemplo, debido al renombramiento de las variables que ocurren en las cláusulas que puede ser infinito. No permitiendo tal situación nada interesante, encontramos tres tipos fundamentales de derivaciones-SLD.

Tipos de derivaciones-SLD

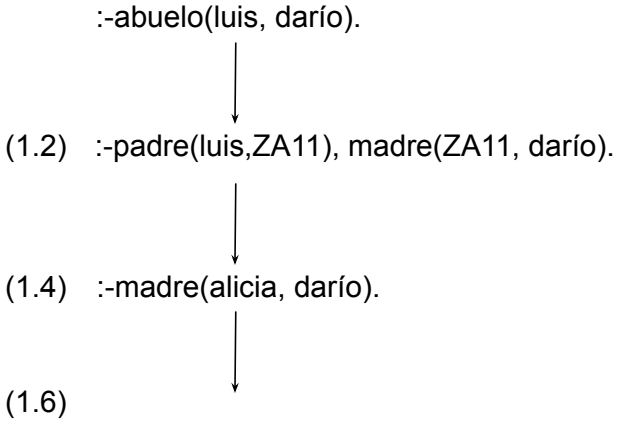
Definición: Una *refutación-SLD* de P ∪ G mediante R es una derivación-SLD finita de la cláusula vacía a partir de P ∪ G mediante R.

Ejemplo 1: Refutación-SLD. Considere el conjunto de cláusulas Familia ∪ {`:- abuelo(luis, darío.)`} previamente descrito. 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.

(1.1) `:- abuelo(luis, darío).`
(1.2) A11: `abuelo(XA11, YA11):- padre(XA11,ZA11),madre(ZA11,YA11).`
$\sigma_1 = \{luis/XA11, darío/YA11\}$ –unificador entre el objetivo (1.1) y la variante A11 (1.2)
(1.3) `:- padre(luis,ZA11),madre(ZA11,darío).` –resolvente de (1.1) y (1.2)
(1.4) P11: `padre(luis, alicia).`
$\sigma_2 = \{alicia/ZA11\}$ –unificador entre (1.4) y el primer sub-objetivo de (1.3)
(1.5) `:-madre(alicia, darío).` –resolvente de (1.3) y (1.4)
(1.6) M11:`madre(alicia, darío).`
$\sigma_3 = \epsilon$ –unificador entre (1.5) y (1.6): cuando no hay variables, el unificador es la sustitución idéntica.
(1.7) –resolvente de (1.5) y (1.6), es la cláusula vacía

Como no hay variables que instanciar en el objetivo el intérprete se limita a dar como respuesta un "si".

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).`

(2.1) `:- abuelo(luis, X).`
(2.2) A11: `abuelo(XA11, YA11):- padre(XA11,ZA11),madre(ZA11,YA11).`
$\sigma_1 = \{luis/XA11, X/YA11\}$ –unificador entre (2.1) y (1.2), recordar que la sustitución X/YA11 se lee: "YA11 será sustituida por X"
(2.3) `:- padre(luis,ZA11),madre(ZA11,X).` –resolvente de (2.1) y (2.2)
(2.4) P11: `padre(luis, alicia).`
$\sigma_2 = \{alicia/ZA11\}$ –unificador entre (2.4) y el primer sub-objetivo de (2.3)
(2.5) `:-madre(alicia, X).` –resolvente de (2.3) y (2.4)
(2.6) M11:`madre(alicia, darío).`
$\sigma_3 = \{darío/X\}$ –unificador entre (2.5) y (2.6)
(2.7)–resolvente de (2.5) y (2.6)

Hallando la composición de los tres umg generados en la derivación se tiene que
$\sigma 1\sigma 2\sigma 3 = \{luis/XA11, darío/YA11, alicia/ZA11, darío/X\}$
denominaremos a tal sustitución la sustitución computada de la derivación-SLD.

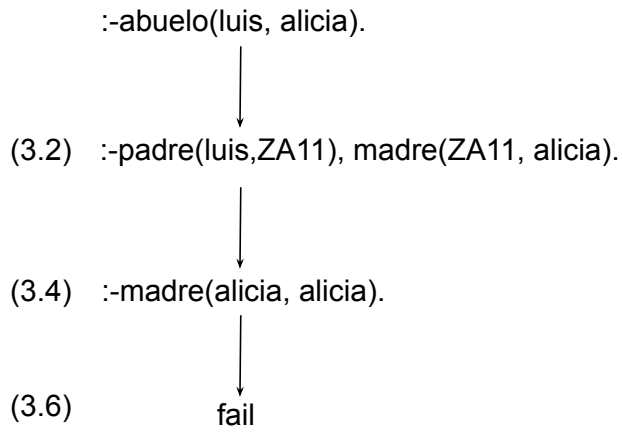
Luego, además de suministrar la deducción de `abuelo(luis, X)`, la derivación-SLD a través del proceso de unificación que se desencadena ofrece un valor para las variables que ocurran en el objetivo G_0, mediante su sustitución computada, en el caso que nos ocupa,$X = darío$.

Derivación-SLD fracasada o 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.

(3.1) `:- abuelo(luis, alicia).`
(3.2) A11: `abuelo(XA11, YA11):- padre(XA11,ZA11),madre(ZA11,YA11).`
$\sigma 1 = \{luis/XA11, alicia/YA11\}$ –unificador entre (3.1) y (3.2)
(3.3) `:- padre(luis,ZA11),madre(ZA11,alicia).` –resolvente de (3.1) y (3.2)
(3.4) P11: `padre(luis, alicia).`
$\sigma 2 = \{alicia/ZA11\}$ –unificador entre (3.4) y el primer sub-objetivo de (3.3)
(3.5) `:-madre(alicia, alicia).` –resolvente de (3.3) y (3.4)
(3.6) fail –no existe una variante de cláusula que unifique con (3.5) 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.

Estrategia de Búsqueda

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.*

Representación: El *conocimiento* necesario para resolver una clase de problemas así como los datos del problema específico 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*.
Veamos a continuación qué respuesta da Prolog a esta dimensión.

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

Como se puede apreciar en los ejemplos de derivaciones desarrollados anteriormente, dado un objetivo recorremos el espacio de búsqueda paso a paso, generando un nuevo objetivo a partir del precedente y de alguna cláusula de programa mediante la aplicación de Resolución-SLD. En estos ejemplos, desarrollados "a mano", hemos hecho uso implícito de alguna estrategia de control que direcciona, orienta el hallazgo de una solución de un objetivo (triunfo, fracaso) y de alguna manera eficiente.

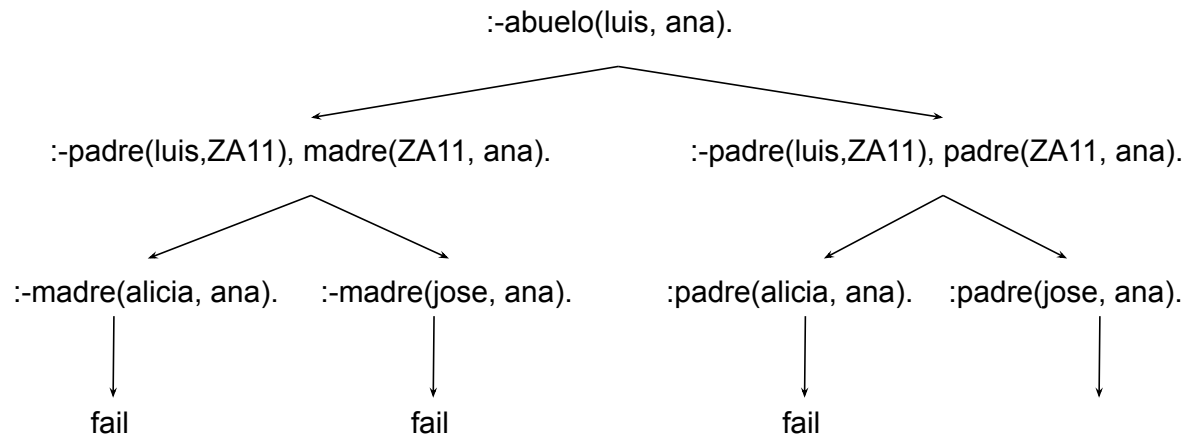
Dado $P \cup G$, lo primero es tener una representación en potencia del espacio de búsqueda del objetivo G. La siguiente definición ofrece una solución a este problema.

Definición: $P \cup G$, 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 $\cup$ {`:- abuelo(luis, ana)`}.



Se puede adoptar, 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.
Ejercicio: construya el árbol de derivación-SLD para Concatena $\cup$ {`:- concatena(X, [1, 2], Y).`}

Dada la forma de árbol que presenta la estructura de un espacio de búsqueda de un objetivo, podemos navegar por dicho espacio generando paulatinamente el árbol de derivación-SLD, utilizando una estrategia de búsqueda en árbol que resulte conveniente. Ejemplo:

- Búsqueda Primero-a-lo-Ancho (BPA) (Breadth-First Search)
- Algoritmo de Búsqueda Primero-en-Profundidad (BPP) (Depth-First Search)

Estrategia de Búsqueda en PROLOG

La estrategia estándar utilizada en PROLOG para general el árbol de búsqueda de un objetivo es una particularización de BPP que aplica conjuntamente la estrategia de búsqueda **retroceso cronológico (chronological backtrack)** (RC), la cual denotaremos por BPP-RC.

Dado $P \cup G$ el control para la búsqueda de ramas triunfos, ramas que terminan en la cláusula vacía, en Prolog es el siguiente:

- Se adopta como regla de selección la que selecciona el sub-objetivo a la extrema izquierda de un objetivo.
- Se toman las cláusulas del programa en un orden fijo, el orden lineal consecutivo en el cual aparecen.
- Se aplica en cada oportunidad una sola cláusula del programa al objetivo seleccionado para generar un nuevo resolvente cuyo primer sub-objetivo se convierte ahora en el nuevo objetivo a resolver.
- Si no es posible expandir el objetivo seleccionado, es decir el objetivo dado es un objetivo fracaso, entonces se retrocede cronológicamente a su objetivo padre el cual pasa a ser el objetivo a resolver utilizando una cláusula no anteriormente aplicada al mismo con la que pueda obtenerse un nuevo resolvente.
- El proceso bien termina, como se ha indicado, al obtenerse una rama triunfo, o porque G es un objetivo fracaso.

O bien transcurre infinitamente sin encontrar una refutación ... habiéndola, como se demuestra con el siguiente programa. $P \cup G$:

```
P:
p(a,b).
p(c,b).
p(X,Y):-p(X,Z),p(Z,Y).
p(X,Y):-p(Y,X).
G:
:- p(a, c).
```

Ejercicio:

1. Muestre que $P \cup G$ tiene una refutación y que todas las cláusulas son necesarias para hallarla.

2. Muestre que cualquiera sea la regla de selección adoptada y cualquiera que sea el orden que se imponga a las cláusulas del programa la estrategia BPP no encuentra la refutación.

Nota: Muestre todo lo anterior construyendo un árbol de derivación.

El orden de las cláusulas y el orden de los sub-objetivos de las cláusulas requieren atención del programador en Prolog. La utilización de BPP-RC obliga a prestar atención al orden en el cual serán tomadas las cláusulas por el intérprete como se demuestra con el programa Concatena ya analizado.