Clase 1.2: Fundamentos Teóricos y Unificación

Profesora: Dra. Yisel Garí

En esta clase se mencionarán algunos fundamentos teóricos a tener en cuenta para la buena comprensión de la base teórica de la programación lógica. Además se definirán los conceptos de cláusula y cláusula definida y se mostrará la utilización de este tipo de cláusula en la programación lógica. Por último se abordará el concepto de unificación, proceso esencial por el que se pueden obtener salidas o soluciones de la ejecución de un programa Prolog.

Fundamentos Teóricos

Definición: Una *lógica de primer orden* LPO consta de un *lenguaje de primer orden* L, y de un sistema lógico asociado.

El lenguaje de primer orden L
Un *lenguaje de primer orden* está constituido por un conjunto de expresiones con las cuales se denotan las entidades de un dominio y se enuncian las propiedades y las relaciones entre las entidades del dominio. Por lo tanto, en su definición debemos considerar las dos dimensiones fundamentales de todo lenguaje denominadas sintaxis y semántica respectivamente. La sintaxis suministra las reglas de formación o gramática de las expresiones aceptadas en un lenguaje, mientras que la semántica suministra las reglas de interpretación en el dominio de tales expresiones.

Sintaxis de L
Sintácticamente L consta de un *alfabeto* y de dos clases de expresiones bien definidas a partir de los símbolos de este alfabeto: *términos* y *fórmulas*.

El alfabeto de L consta de los siguientes conjuntos de símbolos:

- **Var** $= \{x, x_1, ...x_n, ...\}$ donde cada x_i se denomina variable (individual).
- **Func** $= \{f_0/0, f_0/1, ..., f_1/0, f_1/1, ..., f_k/j, ...\}$ donde cada $f_k/j (k, j \geq 0)$ se denomina símbolo de función j-aria (de aridad j). El subíndice k distingue entre diferentes símbolos de funciones con la misma aridad. Un símbolo $f_k/0 \in Func$ se denomina constante (individual) y se denotará por f_k .
- **Rel**$= \{R_0/0, R_0/1, ..., R_1/0, R_1/1, ..., R_k/j, ...\}$ donde cada elemento $R_k/j (k, j \geq 0)$ se denomina símbolo de relación j-aria (de aridad J). Un símbolo $R_k/0 \in R$ se denomina constante proposicional o proposición y se denotará R_k . Los conjuntos **Var**, **Func** y **Rel** son disjuntos y contienen los símbolos no lógicos de L.
- Operadores lógicos, entre los que distinguimos los operadores proposicionales: $\neg/1$ (negación), $\wedge/2$ (conjunción), $\vee/2$ (disyunción), $\Rightarrow /2$ (implicación),$\Leftrightarrow /2$ (bicondicional) y los operadores de cuantificación: $\forall(.)$ (cuantificador universal) y $\exists(.)$ (cuantificador existencial).
- Símbolos auxiliares de escritura: [,] (corchetes), , (coma), (,) (paréntesis).

Términos: La clase de los términos **Term** de una LPO es la clase más pequeña que satisface los siguientes requerimientos:
Sea t una sucesión lineal finita de símbolos del alfabeto de L, entonces:
T1) Si $t \in$ **Var**, entonces $t \in$ **Term**.
T2) Si $f_m/0 \in$ **Func**, entonces $f_m/0 \in$ **Term**.
T3) Si $f_m/n \in$ **Func** y $t_1, ..., t_n \in$ **Term**, entonces $f_m(t_1, ..., t_n) \in$ **Term**.

Fórmulas: La caracterización de la clase de las fórmulas **Form** de una LPO requiere inicialmente la definición de la expresión *fórmula elemental* o *átomo*:

F0) Para $R_m/n \in$ **Rel** y $t_1, ..., t_n \in$ **Term**, $R_m(t_1, ..., t_n)$ es una *fórmula elemental* o *átomo*.

La clase de las fórmulas **Form** de LPO es la clase más pequeña que satisface los siguientes requerimientos:

F1) Si A es una fórmula elemental, entonces $A \in$ **Form**.
F2) Si $A \in$ **Form** , entonces $\neg A \in$ **Form**.
F3) Si A y $B \in$ **Form**, entonces $[A \vee B]$, $[A \wedge B]$, $[A \Rightarrow B]$ y $[A \Leftrightarrow B] \in$ **Form**.
F4) Si $A \in$ **Form**, entonces $\forall(x)A$ y $\exists(x)A \in$ **Form**.

Lo que caracteriza a una LPO como de primer orden es que en esta sólo hay un tipo de variables: las variables individuales que, por la definición de fórmula, pueden aparecer cuantificadas. Una teoría es elemental o de primer orden si es expresable en una LPO.

Ejemplo: A continuación se tienen especificaciones mínimas de una LPO donde se muestra el poder expresivo de su lenguaje para la representación de los términos y fórmulas para expresar relaciones familiares:
Var $= \{x, y, z, ...\}$
Func $= \{ana, luis, luisa, carlos, rosa, el\_padre\_de/1, la\_madre\_de/1\}$
Rel $= \{padre/2, madre/2, abuelo/2\}$
Términos: ana, luis, luisa, carlos, rosa, $el\_padre\_de(carlos)$, $la\_madre\_de(la\_madre\_de(rosa))$
Fórmulas: padre(luis, carlos), padre(luis, luisa), padre(carlos, ana), madre(luisa, rosa).
La siguiente fórmula expresa la definición del concepto abuelo:
$\forall(x)\forall(y)\forall(z)[padre(x, z) \wedge [padre(z, y) \vee madre(z, y)]] \Rightarrow abuelo(x, y)$

Lógica de cláusulas definidas (o cláusulas de Horn[1])

Las cláusulas son fórmulas de uso frecuente en programación lógica y un tipo particular de cláusulas, las cláusulas definidas son las utilizadas para construir un programa en el lenguaje Prolog.
¿Cómo comprender que un programa es un conjunto de fórmulas lógicas? Para dar respuesta a esta pregunta debemos comenzar por las definiciones siguientes, para luego aplicarlas en el programa `concatena(L1, L2, L3)` visto en la primera clase:

```
concatena([], X, X)}.
concatena([X|Rx], Y, [X|Z]) :- concatena(Rx, Y, Z).
```

[1]Se llaman así por el lógico Alfred Horn, el primero en señalar la importancia de estas cláusulas en 1951.

Definición: Una fórmula A se denomina un *literal*, si A es una fórmula elemental o átomo o A es una fórmula ¬B, y B es un átomo, dichos literales se denominan literal positivo y literal negativo respectivamente.
Ejemplos: $x > y, 2 + x > y, \neg x = y, padre(a, b), \neg abuelo(a, b), concatena([], [1, 2], [1, 2])$

Definición: Una fórmula A se denomina *cláusula*, si A tiene la forma $\forall(x_1)...\forall(x_n)(L_1 \vee ... \vee L_m)$ siendo cada L_i un literal (positivo o negativo) y $x_1...x_n$ son las únicas variables que ocurren en $L_1 \vee ... \vee L_m$. Es decir, una cláusula es una disyunción de literales cuyas variables libres están cuantificadas universalmente.

Ejemplos:
$\forall(x)\forall(y)[\neg x > y \vee y < x \vee x = y]$
$\forall(x)\forall(y)\forall(z)(abuelo(x, y) \vee \neg padre(x, z) \vee \neg padre(z, y))$
$\forall(x)\forall(v)\forall(y)\forall(z)(concatena([x|v], y, [x|z]) \vee \neg concatena(v, y, z))$.

De acuerdo con las leyes de la lógica proposicional cualquier cláusula puede ser representada de modo equivalente por
$\forall(x_1)...\forall(x_n)[A_1 \vee ...A_k \vee \neg B_1 \vee ... \vee \neg B_m]$ (Ley conmutativa)
Siendo las A_i (B_j) átomos.
Entonces una cláusula tiene la siguiente representación en forma de implicación:
$\forall(x_1)...\forall(x_n)[A_1 \vee ...A_k \Leftarrow B_1 \wedge ... \wedge B_m]$
al aplicarse las leyes de equivalencias lógicas:
$\neg A \vee \neg B \equiv \neg[A \wedge B]$ (Ley de Morgan) y $\neg A \vee B \equiv A \Rightarrow B$(Definición de la implicación)

Ejemplos:
$\forall(x)\forall(y)[y < x \vee x = y \Leftarrow x > y]$
$\forall(x)\forall(y)\forall(z)[abuelo(x, y) \Leftarrow padre(x, z) \wedge padre(z, y)]$
$\forall(x)\forall(v)\forall(y)\forall(z)[concatena([x|v], y, [x|z]) \Leftarrow concatena(v, y, z)]$

Dado que todas las variables que ocurren en una cláusula están cuantificadas universalmente, podemos hacer una representación implícita de esta cuantificación y eliminar la representación de los cuantificadores:
$y < x \vee x = y \Leftarrow x > y$
$abuelo(x, y) \Leftarrow padre(x, z) \wedge padre(z, y)$
$concatena([x|v], y, [x|z]) \Leftarrow concatena(v, y, z)$

Definición: Una cláusula se denomina *cláusula definida* si la misma contiene a lo sumo un literal positivo. Es decir, una cláusula definida es de la forma $A \Leftarrow B_1 \wedge ... \wedge B_m$
Denominaremos *regla* a tal cláusula definida, siendo A denominada la *cabeza* de la cláusula y $B_1 \wedge ... \wedge B_m$ el *cuerpo* de la cláusula. Si la cláusula tiene sólo cabeza se denomina *hecho* y si tiene sólo cuerpo se denomina *objetivo*. Para diferenciar entre hecho y objetivo, los objetivos comienzan con el símbolo $\Leftarrow$.
Ejemplos:
$abuelo(x, y) \Leftarrow padre(x, z) \wedge padre(z, y)$.
$padre(a, b)$.
$\Leftarrow abuelo(a, x)$.
$concatena([], [2], [2])$.
$\Leftarrow concatena([1], [2], x)$.

Las cláusulas definidas son de uso generalizado en la programación lógica y en particular en Prolog:
Definición: Un programa Prolog es un conjunto de cláusulas definidas.

En lo adelante cuando se hable de un programa sin especificar se estará mencionan- do un programa Prolog.

Los hechos y las reglas de un programa que comienzan con el mismo nombre de predicado o relación constituyen la definición de dicho predicado en el programa.

Ejemplos: Utilizando la notación vista en la primera clase y el programa concatena se pueden formar los siguientes objetivos:

```
:- concatena([1,2,3], [4,5], X).
% Existe X tal que X es la concatenación de [1,2,3] y [4,5]
:- concatena(X, [1,2,3], [1,2,3]).
% Existe la lista X, tal que al concatenarla con [1,2,3] se obtiene la lista [1,2,3]
```

Observe que:

- un programa es propiamente una teoría de primer orden en la lógica de cláusulas definidas constituyendo los hechos y las reglas del programa los axiomas de la teoría.
- un objetivo es un teorema que queremos establecer como consecuencia lógica de un programa.

Se verifica que un objetivo es una cláusula definida de la forma:
$\forall(x_1)...\forall(x_n)[\neg B_1 \vee ... \vee \neg B_m]$
que es lógicamente equivalente a
$\neg\exists(x_1)...\exists(x_n)[B_1 \wedge ... \wedge B_m]$

Por tanto, la lectura lógica del objetivo
`:- concatena([1,2,3], [4,5], X).` es la siguiente:
(a) No existe X tal que X es una lista que sea la concatenación de las listas [1, 2, 3] y [4,5].
Que es la negación de
(b) Existe X, tal que X es una lista que sea la concatenación de las listas [1, 2, 3] y [4,5].
que es el enunciado que se quiere demostrar.

Luego como se estudiará más adelante un programa Prolog es ejecutado por un *intérprete* del lenguaje que a partir de la definición dada del predicado concatenar en el programa, tratará de *demostrar* (lo que (b) plantea) *refutando* (con lo que (a) plantea), es decir, un intérprete de Prolog será desde un punto de vista lógico un *demostrador por refutación* de teoremas.

Sabemos por lógica proposicional que una demostración por refutación llega en su paso más definitivo a una fórmula contradictoria o contradicción. En la lógica de cláusulas la *cláusula vacía* denotada por $\cdot$ representará esta contradicción.

Observe sin embargo en el ejemplo que nos ocupa que el intérprete debe hacer más que una prueba por refutación: simultáneamente con la refutación del objetivo debe *suministrar un valor* para X, precisamente la concatenación de las listas [1, 2, 3] y [4, 5]. Esto último es muy

importante: en lógica o en demostración automática puede que sólo nos interese la refutación, es decir, la demostración del teorema, pero en un programa lógico lo que más nos interesa es el valor de respuesta o *salida* que el programa daría a las variables que ocurren en un objetivo.

La unificación

Ya se ha revelado que la programación lógica más que deducir objetivos está interesada en lo que realmente la ayuda a establecerse como modelo de computación: asociar valores a las variables de los objetivos suministrándolos como salida de ejecución de un programa.

Las variables de un programa lógico son variables lógicas por ello hablamos de asociar valores a las variables o de instanciarlas y no de asignación, término que deberá evitarse por hacer referencia en la programación estándar (imperativa) a la operación asociada a las denominadas proposiciones de asignación mediante la cual se asignan valores a las variables de un programa. Como se verá más adelante, una vez instanciada una variable lógica no puede ser instanciada nuevamente, mientras que la asignación es destructiva: una variable en un programa imperativo puede cambiar varias veces de valor mediante asignaciones sucesivas. Por supuesto ambas formas de asociar valores a variables son igualmente valiosas en la programación.

En la PL los valores con los que se puede instanciar una variable son términos cualesquiera del lenguaje de un programa. La instanciación se basa en la operación lógica de sustitución (de una variable por un término) que constituye la operación fundamental de un proceso denominado unificación, a cuyo estudio dedicaremos la presente sección.

Definición: Una sustitución σ es un conjunto finito de sustituciones $t_1/v_1, ..., t_n/v_n$ donde v_i es una variable y t_i es un término, tales que $t_i \neq v_i$ y $v_i \neq v_j$ para $i \neq j$.
Observaciones:

- t_i/v_i se lee: "sustitución de todas las ocurrencias de la variable v_i por el término t_i ".
- Denotaremos por ϵ la sustitución idéntica.

Ejemplos: $\sigma_1 = \{2/X, W/Y\}, \sigma_2 = \{g(Z)/X, a/Y\}, \sigma_3 = \{c/X, a/Y\}$

Definición: Una sustitución $\sigma = t1/v1, ..., tn/vn$ es básica si $t_i (i = 1, ..., n)$ es un término básico.

Ejemplos: $\sigma = \{f(c)/X, a/Y\}$

Si C es la cláusula $R(X, f(Y), b)$ entonces la aplicación de la sustitución $\sigma = 2/X, W/Y$ a C da como resultado la cláusula $R(2, f(W), b)$ que denotaremos por Cσ.

Definición: (Composición de sustituciones): Sean σ y σ' dos sustituciones, la composición $\sigma\sigma'$, también denotada por $\sigma(\sigma')$ es la sustitución que se obtiene

1. aplicando σ' a los términos que aparecen en las sustituciones de σ
2. adicionando al conjunto resultante los elementos de σ' cuyas variables no ocurren entre las variables de σ y

3. suprimiendo toda sustitución resultante t_i/v_i con $t_i = v_i$

Ejemplo: Sean las sustituciones $\sigma = \{g(X,Y)/Z\}$, $\sigma' = \{a/X, b/Y, c/W, d/Z\}$, entonces $\sigma\sigma' = \{g(a,b)/Z, a/X, b/Y, c/W\}$
Propiedades de la composición de sustituciones:

1. asociativa: $\sigma(\sigma'\sigma'') = (\sigma\sigma')\sigma''$
2. Para toda $\sigma, \sigma\epsilon = \epsilon\sigma$
3. La composición de sustituciones no es conmutativa.

Definición: Denominaremos expresión simple a todo literal o término.

Sea E un conjunto finito de expresiones simples y σ una sustitución, entonces Eσ es el conjunto de las expresiones que se obtienen al aplicar la sustitución σ a cada expresión simple de E.

Ejemplo: Dados $E = \{p(X,Y), p(f(a),Z), p(f(Z),Y)\}$ y $\sigma = \{f(a)/X, a/Y, c/Z\}$ se tiene
$E\sigma = \{p(f(a),a), p(f(a),c), p(f(c),a)\}$

Definición: Sea E un conjunto finito de expresiones simples y σ una sustitución, σ es un *unificador* de E si y sólo si Eσ es un conjunto unitario.

Definición: Un conjunto finito de expresiones simples E es unificable si existe un unificador de E.

Ejemplo: Para $E = \{p(X,Y), p(f(a),Z), p(f(Z),Y)\}$ se tiene que $\sigma = \{f(a)/X, a/Y, a/Z\}$ es un unificador de E, dado que $E\sigma = \{p(f(a),a)\}$.

Definición: σ es un unificador más general (umg) del conjunto de expresiones E si y sólo si para toda γ tal que γ es unificador de E, existe una sustitución σ' tal que $E\gamma = (E\sigma)\sigma'$.
Ejemplo:
Sea $E = \{p(f(X),Z), p(Y,a)\}$
Luego $\gamma = \{f(a)/Y, a/Z, a/X\}$ es un unificador de E.
Y $\sigma = \{f(X)/Y, a/Z\}$ es un umg de E ya que existe $\sigma' = \{a/X\}$ tal que $E\gamma = (E\sigma)\sigma'$

Se verifica fácilmente que si σ y σ' son umgs de E, entonces $E\sigma = E\sigma'$ con un renombramiento adecuado de las variables.

Ejemplo:
Sea $E = \{p(f(X),Z), p(f(Y),V)\}$
Luego $\sigma = \{Y/X, V/Z\}$ y $\sigma' = \{X/Y, V/Z\}$ son umgs de E siendo $E\sigma = E\sigma'$ con el renombramiento de variables {X/Y} o {Y/X}.

Algoritmo de Unificación de Robinson (AUR)

Con objeto de encontrar el unificador de máxima generalidad de dos términos se han propuesto numerosos algoritmos siendo el más conocido el de Robinson [2] (1965).

[2] John Alan Robinson, filósofo, matemático y científico de la computación (1930-2016).

Entre los elementos que vamos a considerar se encuentran el llamado primer par de discordancia entre dos términos, y la aparición de variables dentro de un término *occur check* (comprobación de apariciones).
Primer par de discordancia: Tomemos dos términos que no sean iguales. Eso significa que habrá alguna diferencia entre ellos. Esta diferencia puede hallarse en el nombre del funtor, en el número de argumentos, o en alguno de los argumentos.
Si la diferencia está en el nombre del funtor, el primer par de discordancia es la pareja formada por los dos términos. Si está en el número de argumentos, al igual que antes, el primer par de discordancia resulta ser el formado por los dos términos considerados.
En cambio, si coinciden en los funtores y en el número de argumentos, el primer par de discordancia debe ser buscado entre sus argumentos empezando a considerar estos de izquierda a derecha. Se busca en el primer argumento, y si aquí no lo hubiera, se busca en el segundo, . . .
Es evidente que al final se debe encontrar este par de discordancia pues hemos partido de dos términos distintos.

Ejemplo: Sean los términos $p(a, f(b, c), g(X)), p(a, f(X, Y), g(h(Z))$
Conjunto desacuerdo $D = \{b, X\}$

Ejemplo: Sean los términos $E = \{p(g(a, b, c), X), p(f(X, a), Z)\}$
Conjunto desacuerdo $D = \{g(a, b, c), f(X, a)\}$

La comprobacion de apariciones: Con la comprobación de apariciones (*occur check*) simplemente queremos detectar cuándo una variable aparece dentro de un término. Si bien desde un punto de vista formal esto no ofrece ninguna complicación, sí la ofrece cuando se intenta sistematizar este procedimiento de tener que buscar por cada argumento y dentro del mismo la aparición de una variable. En cualquier caso, sólo se trata de eso.

Ejemplo: La variable X se encuentra dentro de los términos $p(a, f(b, X))$ y $q(X, g(X, Y))$ pero no de los términos $p(a, Z, g(a, b)yq(a, b)$.

Instrucciones de AUR:

Sean E y F dos términos que queremos unificar. Consideramos inicialmente $\sigma_0 = \{\}$ una sustitución vacía, es decir, que no cambia ninguna variable. Dado que vamos a realizar un proceso iterativo, consideramos inicialmente $E_0 = \sigma_0(E)$ y $F_0 = \sigma_0(F)$. En cada iteración k del algoritmo se realizan los siguientes pasos:

Paso 1 Si $E_k = F_k$ entonces las cláusulas E y F son unificables y un unificador de máxima generalidad es $\sigma = \sigma_k ... \sigma_0$. Además, el término E_k es el término unificado. En este caso el proceso termina aquí.

Paso 2 Si $E_k \neq F_k$ entonces se busca el primer par de discordancia entre E_k y F_k. Sea este Dk.

Paso 3 Si Dk contiene una variable y un término (pueden ser dos variables y una de ellas hace de término) pasamos al siguiente paso. En otro caso los términos no son unificables y terminamos el proceso.

Paso 4 Si la variable aparece en el término (*occur check*) E y F no unifican y terminamos. Si esto no ocurre pasamos al siguiente paso.

Paso 5 Construimos una nueva sustitución que vincule la variable con el término de Dk. Sea esta sustitución $\sigma_k + 1$. Construimos ahora dos nuevos términos $E_k + 1 = \sigma_k + 1(Ek)$ y $F_k + 1 = \sigma_k + 1(Fk)$ y volvemos al paso 1.

Este algoritmo siempre termina para dos términos cualesquiera. Si los términos no eran unificables terminará indicandolo así y si eran unificables devolverá un unificador de máxima generalidad y el término resultante unificado.

Ejemplo:
Sean los términos $p(a, X)$ y $p(X, Y)$ aplicando el algoritmo paso a paso tenemos:
$E_0 = p(a, X)$, $F_0 = p(X, Y)$ y $\sigma_0 = \{\}$

Iteración 1 Desarrollamos para $k = 0$
Paso 1 Como $E_0 \neq F_0$ vamos al paso 2.
Paso 2 $D_0 = \{a, X\}$. Vamos al paso 3.
Paso 3 En D_0 uno es un término (a) y el otro una variable (X). Vamos al paso 4.
Paso 4 La variable X no aparece en el término a. Vamos al paso 5.
Paso 5 Sea $\sigma_1 = \{a/X\}$. Sean también $E_1 = \sigma_1(E_0) = p(a, a)$ y $F_1 = \sigma_1(F_0) = p(a, Y)$. Volvemos al paso 1.

Iteración 2 Desarrollamos para $k = 1$
Paso 1 Como $E_1 \neq F_1$ vamos al paso 2.
Paso 2 $D_1 = \{a, Y\}$. Vamos al paso 3.
Paso 3 En D_1 uno es un término (a) y el otro una variable (X). Vamos al paso 4.
Paso 4 La variable Y no aparece en el término a. Vamos al paso 5.
Paso 5 Sea $\sigma_2 = \{a/Y\}$. Sean también $E_2 = \sigma_2(E_1) = p(a, a)$ y $F_2 = \sigma_2(F_1) = p(a, a)$. Vamos al paso 1.

Iteración 3 Desarrollamos para $k = 2$
Paso 1 Como $E_2 = F_2$ el algoritmo termina con $\sigma = \sigma_2\sigma_1\sigma_0 = \{a/X, a/Y\}$ como unificador de máxima generalidad.

Algoritmo de unificacion en Prolog
El algoritmo de unificación que utiliza Prolog es básicamente igual al aquí descrito excepto que no dispone del Paso 4. Esto quiere decir que no verifica el *occur check*. La razón de prescindir de este paso es por cuestión de eficiencia ya que la comprobación de *occur check* consume demasiado tiempo, y hay que realizarla muchas veces. Por otro lado, no es habitual encontrar situaciones que deriven en programas donde se produzcan *occur checks*. En cualquier caso, si se presenta, dependiendo de la implementación de Prolog, se comporta de una manera u otra. Hay sistemas Prolog que lo detectan y avisan del hecho, y otros que simplemente dejan de funcionar.