Conferencia 2: Programación Declarativa

M.Sc. Dafne García de Armas

Curso 2018-2019

Tema: Cláusulas definidas. Algoritmo de unificación.

La programación lógica surgió propiciada por las investigaciones que se realizaron en el área de la demostración automática de teoremas, en particular, por resultados obtenidos para demostradores automáticos de teoremas que usan la Lógica de Predicados (Lógica de Primer Orden (LPO)) y el procedimiento de prueba por refutación (reducción al absurdo) como método de demostración.

En esta conferencia 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 mostrarán el algoritmo de unificación, proceso esencial por el que se pueden obtener salidas o soluciones de la ejecución de un programa Prolog.

LÓGICA DE PREDICADOS

La programación lógica se basa en la lógica de predicados, también denominada lógica de primer orden. Recordemos cada una de sus componentes.

El lenguaje de la Lógica de Predicados o lenguaje de primer orden L

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.

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:

i) ***Variables:*** *{ x, x1, ... xn, ...}* donde cada x_i se denomina *variable (individual)*.

ii) ***Constantes:*** π, e, # (puede ser considerado también una función de aridad 0)

iii) ***Operadores lógicos:*** entre los que distinguimos los *operadores proposicionales*: ¬/1 (*negación)*, ∧/2 (*conjunción)*, ∨/2 (*disyunción)*, ⇒/2 (*implicación)*, ⇔/2 (*bicondicional)* y los *operadores de cuantificación*: ∀(.) (cuantificador *universal*) y ∃(.) (*cuantificador existencial*).

iv) ***Funciones:*** *{* $f_0/0, f_0/1,..., f_1/0, f_1/1,..., f_k/j,...$ *}* donde cada f_k/j *(k, j ≥ 0)* se denomina *símbolo de función j-aria (de aridad j)*. El subíndice *k* distingue entre diferentes símbolos de funciones con la misma aridad.

v) ***Relaciones:*** *{* $R_0/0, R_0/1, ..., R_1/0, R_1/1,..., R_k/j, ...$ *}* donde cada elemento R_k/j *(k, j ≥ 0)*se denomina *símbolo de relación j-aria (de aridad J)*. Un símbolo $R_k/0 \in \boldsymbol{R}$ se denomina *constante proposicional* o *proposición* y se denotará R_k.

vi) **Signos de agrupación:** [,], { , }, (,)

Términos: La clase de los *términos* 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:

i) Toda constante individual o variable es un término.
ii) Si $t_1, t_2, \dots, t_n$ son términos y f_n *es un símbolo de función n-aria entonces* $f(t_1, t_2, \dots, t_n)$ *es un término.*
iii) No existen términos que no sean definidos por las reglas i y ii

Fórmulas: Son secuencias de símbolos del lenguaje que se consideran sintácticamente correctas denominadas fórmulas bien formadas. La caracterización de la clase de las fórmulas de una LPO requiere inicialmente la definición de la expresión *fórmula elemental* o *átomo*:

Fórmula Elemental o Átomo: Sean $t_1, t_2, \dots, t_n$ términos y sea r un símbolo de relación n-aria entonces $r(t_1, t_2, \dots, t_n)$ es una *fórmula elemental* o *átomo*.

Expresiones que son fórmulas:

i. Si *A* es una fórmula elemental, entonces *A es una fórmula*.
ii. Si *A* es una fórmula, entonces ¬*A* es una fórmula.
iii. Si *A* y *B* son fórmulas, entonces *[A∨B], [A∧B], [A⇒B] y [A⇔B]* son fórmulas.
iv. Si *A* es una fórmula donde x ocurre libre, entonces ∀*(x)A* y ∃*(x)A* son fórmulas
v. No existen fórmulas que no sean definidas por las reglas anteriores.

Lo que caracteriza a una LPO como *de primer orden* es que en este 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:

Variables *= {x, y, z, ... }*

Constantes *= { ana, luis, luisa, carlos, rosa }*

Funciones *= {el_padre_de/1, la_madre_de/1 }*

Relaciones *= {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:

∀(x)∀(y)∀(z)[padre(x, z)∧[padre(z, y)∨ madre(z, y)]]⇒ abuelo(x, y)

LÓGICA DE CLÁUSULAS DEFINIDAS

En esta conferencia restringiremos nuestra atención a Teorías Elementales (Teorías de Primer Orden) cuyas fórmulas son denominadas *cláusulas.* 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.

Literal: Si A es una fórmula elemental o átomo entonces A y ¬A son literales, dichos literales se denominan literales positivos y negativos respectivamente.

Cláusula: Una fórmula A se denomina cláusula, si A tiene la forma:

$$\forall(X1)\forall(X2)\ldots\forall(X)\ (L1 \vee L2 \vee \ldots \vee Ln)$$

siendo cada L_i un literal (positivo o negativo) y $x_1 \ldots x_n$ son las únicas variables que ocurren en $L_1 \vee \ldots \vee L_n$. 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) ∨ ¬padre(x, z) ∨ ¬padre(z, y)).
$\forall(x)\ \forall(v)\forall(y)\ \forall(z)$ (concatena([x|v], y,[x|z]) ∨ ¬concatena(v, y, z)).

De acuerdo con las leyes de la lógica proposicional cualquier cláusula puede ser representada como:

∀(x1)… ∀(xn)[A1 ∨ … Ak ∨ ¬B1 ∨ … ∨ ¬Bm]
(Ley conmutativa) Siendo las Ai (Bj) literales positivos.

Y expresándola como una implicación se tiene que:
∀(x1)… ∀ (xn)[A1 ∨ … Ak ⇐ B1 ∧ … ∧ Bm]

Esto se alcanza aplicando las leyes de equivalencias lógicas:

¬A ∨ ¬B ≅ ¬[A ∧ B] (Ley de Morgan) y $\neg A \vee B \cong A \Rightarrow B$ (Definición de la implicación)

Ejemplos:

$\forall(x)\ \forall(y)[\ y < x \vee x = y \Leftarrow x > y]$

∀(x) ∀(y) ∀(z) [abuelo(x, y) ⇐ padre(x, z) ∧ padre(z, y)].

∀(x) ∀(v) ∀(y) ∀(z) [concatena([x|v], y,[x|z]) ⇐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) ⇐ padre(x, z) ∧ padre(z, y).

concatena([x|v], y,[x|z]) ⇐concatena(v, y, z).

Cláusula definida: 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 ⇐ B1 ∧ … ∧ Bm.

Programa Prolog: es un **conjunto de cláusulas definidas**.

Regla: es una cláusula definida en la cual A es la cabeza de la regla y a B1 ∧ … ∧ Bm es el cuerpo de la regla.

Objetivo: es una cláusula definida que no tiene cabeza.

Hecho: es una cláusula definida que no tiene cuerpo.

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.

Interpretación de los objetivos y lectura declarativa de los objetivos

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 utilizando para ello reducción al absurdo.

Se verifica que un objetivo es una cláusula definida de la forma:

$$\forall(x1) \ldots \forall(xn)[\neg B1 \vee \ldots \vee \neg Bm] \cong \neg\exists (x1) \ldots \exists (xn)[B1 \wedge \ldots \wedge Bm]$$

Ejemplo:

:- concatena([1,2,3], [4,5], X).

% No Existe X tal que X sea el resultado de concatenar [1,2,3] y [4,5]

:- concatena(X, [1,2,3], [1,2,3]).

% No Existe X, tal que al concatenarla con [1,2,3] se obtiene la lista [1,2,3]

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 □ 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]. Por lo tanto, aunque en una demostración automática puede que sólo nos interese la refutación, es decir, la demostración del teorema, 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. ALGORITMO DE UNIFICACIÓN

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, ya que una vez instanciada una variable lógica esta 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 Programación Lógica los valores con los que se puede instanciar una variable son términos cualesquiera del lenguaje de un programa. **La operación que permite instanciar una variable lógica es la sustitución** (de una variable por un término) que constituye la **operación fundamental** de un proceso denominado ***unificación***, y se realiza sustituyendo donde quiera que esté la variable por un término del lenguaje de un programa.

Definición: Una sustitución σ es un conjunto finito de sustituciones $\{ t1/v1, \ldots, tn/vn \}$ 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.

Ejemplo:
Si C es la cláusula R(X, f(Y), b) y σ = {2/X, W/Y} entonces la aplicación de la sustitución σ a C da como resultado la cláusula R(2, f(W), b) que denotaremos por Cσ.

Definición: Dada la cláusula *C*, se dice que *C′* es un *ejemplo* de *C* si existe una sustitución σ tal que *C′ = Cσ*. Si *C′* no tiene ocurrencia alguna de variables, entonces *C′* es un *ejemplo básico* de *C*.

Composición de sustituciones: Sean σ y σ' dos sustituciones, la composición σσ´, también denotada por σ(σ') es la sustitución que se obtiene

a) aplicando σ' a los términos que aparecen en las sustituciones de σ,
b) adicionando al conjunto resultante los elementos de σ' cuyas variables no ocurren entre las variables de σ y
c) suprimiendo toda sustitución resultante t_i/v_i con $t_i = v_i$

Ejemplo: Sean las sustituciones σ = {g(X, Y)/ Z}, σ' = {a/X, b/Y, c/W}, entonces

σσ' = {g(a, b)/Z, a/X, b/Y, c/W}

Propiedades de la composición de sustituciones:

1) *asociativa: σ(σ′σ″)=(σσ′)σ″* Demostración: Ejercicio.

2) Para toda *σ*, *σε = εσ* Demostración: ε es la sustitución idéntica por definición.

La composición de sustituciones no es conmutativa, como se puede verificar mediante un contraejemplo.

Definición: Denominaremos *expresión simple* a todo literal o término.

Instancia de E: Sea E un conjunto finito de expresiones simples (literales o términos) 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

$E\sigma = \{p(f(a), a), p(f(a), c), p(f(c), a)\}$

Unificador de E: 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 $E\sigma=\{p(f(a), a)\}$.

Unificador más general: σ es un umg del conjunto de expresiones E si y sólo si para cualquier unificador θ de E se que cumple que existe una sustitución γ tal que

$$\theta = \sigma\gamma$$

Es decir **todo unificador de E se puede obtener a partir de la composición de una sustitución con el umg de E.**

Un unificador θ para listas de literales L1 y L2 se dice más general que un unificador θ1 cuando existe otro unificador θ2 tal que θθ2 = θ1. Por ejemplo, si queremos unificar p(X, Y, b) con p(X, a, Z), el unificador θ = {Y = a, Z = b} es más general que el unificador ´ θ1 = {X = c, Y = a, Z = b}, puesto que θ{X = c} = θ1.

Algoritmos de Unificación

Definición: Una *ecuación* es una expresión de la forma $s = t$ donde s y t son términos del lenguaje de una teoría $\boldsymbol{T}$ y = denota la relación de igualdad.

Definición: Una *solución* σ de un conjunto de ecuaciones $E = \{s_1 = t_1, ..., s_n = t_n\}$ es una sustitución para el conjunto de las variables que ocurren en E tal que $s_1\sigma = t_1\sigma, ..., s_n\sigma = t_n\sigma$.

El conjunto de soluciones de un conjunto de ecuaciones es denotado por *sol(E)*.

Ejemplo: Para $E = \{ f(X, Z) = f(Y, g(Y)) \}$ son soluciones:

(1) = *{ g(a)/X, g(a)/Y, g(g(a))/Z }*

(2) = *{ a/X, a/Y, g(a)/Z }*

(3) = *{ Y/X, g(Y)/Z }*

Luego, (1), (2), (3) ∈ *sol(E)*.

Definición: Un conjunto de ecuaciones *está resuelto* (*es una forma resuelta*) si tiene la forma

$\{ v_1 = t_1, ..., v_n = t_n \}$ donde las v_i $(1<i<n)$ son variables distintas, las cuales no ocurren en la parte derecha de ninguna ecuación del conjunto.

Ejemplo: *E = { X = a, Y = f(a)}* está resuelto.

Definición: Dados los conjuntos de ecuaciones *E* y *E′*, *E = E′* si y sólo si *sol(E)=sol(E')*.

Ejemplo: Para *E = {f(X, g(a)) = f(a, Z)} y E′ = {g(X) = Z, f(X) = f(a)}* se tiene *sol(E) = {a/X, g(a)/Z} = sol(E')*, luego *E = E′*.

Algoritmo de Unificación Ecuacional (AUE)

Dado un conjunto de ecuaciones E, transformar a E en un conjunto de ecuaciones en forma resuelta E' tal que E y E' son conjuntos de ecuaciones equivalentes.

Instrucciones de AUE:

- (AUE1) Reemplazar toda ecuación de la forma: $f(s1,\ldots,sn) = f(t1,\ldots,tn)$ por las ecuaciones: $s1 = t1,\ldots,sn = tn$.
- (AUE2) Si existe alguna ecuación de la forma: $f(s1,\ldots,sn) = g(t1,\ldots,tm)$ siendo f ≠ g o n ≠ m entonces parar: fracaso.
- (AUE3) Eliminar toda ecuación de la forma x = x
- (AUE4) Reemplazar toda ecuación de la forma t=x donde t no es una variable, por la ecuación x = t.
- (AUE5) Si existe alguna ecuación de la forma x = t y x tiene otra ocurrencia en el conjunto de las ecuaciones, entonces:

  (i) si x ocurre en t, entonces parar: fracaso, /*chequeo de ocurrencia de lo contrario
  (ii) aplicar la sustitución {t/x} en el resto de las ecuaciones.

Las instrucciones del algoritmo se aplican siempre que sea posible sin ningún orden particular. El algoritmo termina cuando no se pueda aplicar ninguna instrucción o cuando pare por fracaso.

Ejemplos:

(1) Sea E = { p(X, f(c)) = p(Z, f(X)) }

E1={ X = Z, f(c) = f(X) } por (AUE1)

E2 = { X = Z, c = X } por (AUE1)

E3 = { X = Z, X = c } por (AUE4)

E4={ X = Z, Z = c } por (AUE5)

E5={ X = c, Z = c } por (AUE5)

Luego, en este primer caso, la transformación E´ de E a la que hemos llegado está en forma resuelta y es un umg de E siendo igual a { c/X, c/Z }.

(2) Sea E = { q(X, X) = q(Y, g(Y)) }

E1={ X = Y, X = g(Y) } por (AUE1)

E2={ X = Y, Y = g(Y) } por (AUE5)

FRACASO Y ocurre en g(Y) por (AUE5)

Análisis de la terminación del algoritmo:

- (AUE1) disminuye el número de ocurrencia de símbolos funcionales en el conjunto de ecuaciones a resolver y como este número es finito, (AUE1) se aplica un número finito de veces.

- (AUE2) se aplica una sola vez e indica terminación (con fracaso).

- (AUE3) y (AUE4) disminuyen el número total de ocurrencia de variables y símbolos funcionales en la parte izquierda de las ecuaciones. Luego (AUE3) y (AUE4) se aplican un número finito de veces.

- La aplicación de (AUE5) termina con fracaso o elimina todas las ocurrencias de una variable en la parte derecha de las ecuaciones. Para una variable dada, (AUE5) puede aplicarse sólo una vez y como el número de variables es finito en el conjunto de ecuaciones, en total (AUE5) sólo puede aplicarse un número finito de veces.

Por tanto, el algoritmo termina para cualquier entrada E.

Análisis de la corrección del algoritmo:

AUE es correcto si al ser aplicado a un conjunto de ecuaciones E:

a) Si E tiene solución, AUE para suministrando un conjunto de ecuaciones en forma resuelta E' tal que, $E' = E$.

b) Si E no tiene solución, AUE para señalando fracaso.

Verificación de a):

$E' = E$: Basta verificar que las instrucciones (AUE1), (AUE3), (AUE4), (AUE5)(ii) de ser posible su aplicación transforman un conjunto de ecuaciones en otro conjunto de ecuaciones equivalente, es decir que tiene las mismas soluciones que el anterior (ejercicio).

E' es una forma resuelta de E: En E' todas las partes izquierdas de las ecuaciones son variables, de lo contrario (AUE1), (AUE3) o (AUE4) pudieran ser aplicados. Además, todas estas variables son distintas y no aparecen en las partes derechas de las ecuaciones, de lo contrario (AUE5)(ii) pudiera ser aplicado.

Verificación de b):

Si AUE termina con fracaso entonces se ha aplicado (AUE2) o (AUE5)(i), en cualquiera de los casos se ha encontrado una ecuación $s_i = t_i$ para la cual no es posible obtener una sustitución básica σ tal que $s_i\sigma = t_i\sigma$. Es decir, el conjunto de ecuaciones en el paso fracaso no tiene solución y como este conjunto es equivalente a E, entonces E tampoco tiene solución.

Tarea: Investigar acerca del **Algoritmo de Unificación de Robinson (AUR)**

Ejemplo:

En los casos del Inspector Craig, siempre se plantean proposiciones que se saben que son verdaderas y a partir de ellas hay que descubrir al o a los culpables. En cierto caso, cuatro defendidos A, B, C y D estaban involucrados y se establecieron las cuatro afirmaciones siguientes:

- Si A y B son culpables entonces C es culpable.
- Si A es culpable entonces al menos uno entre B y C es culpable.
- Si C es culpable entonces D también es culpable.
- Si A es inocente entonces D es culpable.

a. Formule declarativamente las proposiciones anteriores en Prolog.

%Si A y B son culpables entonces C es culpable.

p1(c,c,c,_).

p1(c,i,_, _).

p1(i,c,_, _).

p1(i,i,_, _).

%Si A es culpable entonces al menos uno entre B y C es culpable

p2(c, c, _, _).

p2(c,_,c, _).

p2(i, _, _, _).

%Si A es inocente entonces D es culpable.

p4(i,_,_, c).

p4(c,_,_,_).

%Si C es culpable entonces D también es culpable.

p3(_,_,c, c).

p3(_,_,i, _).

b. Declare el predicado solve(A, B, C, D) que triunfa si las proposiciones anteriores son verdaderas atendiendo a la inocencia o culpabilidad de los sospechosos.

Ejemplo:

:-solve(culpable, B, C, culpable)

B = C, C = culpable

solve(A, B, C, D):- p1(A,B,C,D), p2(A,B,C,D), p3(A,B,C,D), p4(A,B,C,D).

c. Suponga que el Inspector Craig cree que D es culpable, formule un objetivo que permita corroborar o refutar la suposición del inspector Craig. Justifique su respuesta.

solve(_,_,_,c)