Lógica Computacional, 2025-2

Unidad 3: Lógica de Predicados

Sintaxis

Manuel Soto Romero

Luis Fernando Loyola Cruz

26 de abril de 2025
Facultad de Ciencias UNAM

Hemos utilizado fórmulas de la Lógica Proposicional siempre que representamos proposiciones en español, es decir, enunciados que pueden ser verdaderos o falsos. Consideremos los siguientes ejemplos:

- ★ Todos los trenes son eléctricos.
- ★ Algunos peces viven en agua dulce.
- ★ Ningún automóvil vuela.
- ★ Todos los estudiantes llevan mochila.
- ★ La Península de Yucatán es la única región de México rodeada de agua por tres lados.

Podemos observar que cada uno de estos enunciados es una proposición, ya que posee un valor de verdad bien definido. No obstante, se diferencia de las proposiciones analizadas previamente, pues en ellos no aparecen explícitamente los conectivos lógicos estudiados. Por esta razón, la única forma de formalizarlos hasta el momento es asignándoles una única variable proposicional que represente el enunciado completo.

Por otro lado, para analizar argumentos correctos, no basta con un lenguaje lógico que nos permita representar proposiciones aisladas. También es fundamental contar con un sistema lógico más expresivo que nos permita capturar la estructura interna de los enunciados y la relación entre ellos. La Lógica Proposicional, aunque útil en muchos casos, no siempre es suficiente para evaluar correctamente la corrección de un argumento. Veamos un ejemplo.

Ejemplo 1

Consideremos el siguiente argumento lógico:

- ★ Algunas ciudades tienen un sistema de transporte subterráneo.
- ★ Todas las ciudades con un sistema de transporte subterráneo cuentan con una red de autobuses.
- ★ Por lo tanto, algunas ciudades cuentan con una red de autobuses.

Nuestra intuición nos dice que este argumento es correcto. Sin embargo, si intentamos representarlo en Lógica Proposicional, obtenemos una estructura similar a:

$$p, q / \therefore r$$

Bajo las técnicas estudiadas ahora, esta representación no garantiza la correctez del argumento. En Lógica Proposicional, la conclusión no se sigue necesariamente de las premisas, lo que sugiere que el

argumento es incorrecto dentro de este sistema. No obstante, esto no significa que el argumento sea incorrecto en términos generales; simplemente indica que la Lógica Proposicional no tiene suficiente poder expresivo para capturar la relación lógica subyacente.

Para abordar este tipo de razonamientos, necesitamos un lenguaje de especificación más potente pues el lenguaje de la Lógica Proposicional que hemos estudiado hasta ahora no posee la capacidad expresiva suficiente para analizar proposiciones y argumentos que incluyen referencias a conjuntos de objetos, como ocurre en los ejemplos anteriores. Esta necesidad nos lleva al estudio de la **Lógica de Primer Orden**, también conocida como **Lógica de Predicados**, que nos permitirá representar y analizar argumentos de manera más precisa y rigurosa.

Para adentrarnos en la Lógica de Predicados, comenzaremos estudiando su **sintaxis**, es decir, las reglas que definen como se construyen correctamente las expresiones en este sistema lógico.

1. Introducción

Empezaremos por definir formalmente el **lenguaje de la lógica de predicados** al que llamaremos LPred. A diferencia de la Lógica Proposicional, este lenguaje depende de un **universo de discurso** específico y de las propiedades que queremos describir sobre los elementos que lo conforman.

Ejemplo 2

La oración:

El árbol en el jardín es el más alto de la cuadra.

Puede ser verdadera en la casa de la familia López, pero falsa en la de la familia Ramírez. Este contexto en el que evaluamos la veracidad de una afirmación es lo que llamamos universo de discurso. Es decir, el conjunto de elementos sobre los cuales se interpretan los enunciados.

A continuación, exploraremos los elementos que conforman este lenguaje.

Términos

Los términos constituyen la categoría sintáctica que representa a los individuos dentro del universo de discurso.

Definición 1. (Término)

Un **término** puede ser una **constante**, una **variable**, o un **símbolo de función** aplicado a uno o más términos.

Ejemplo 3

Supongamos que nuestro universo de discurso está formado por ciudades del mundo.

★ Las variables x y y representan cualquier ciudad.

- ★ La constante p denota París, y la constante t a Tokio.
- ★ El símbolo de función pais , de aridad^a 1, representa la operación que recibe una ciudad y devuelve el país en el que se encuentra. Es decir $\text{pais}(x)$ denota al país al que pertenece x .

Dado que cada ciudad pertenece a un único país, esta asociación es funcional. En particular $\text{pais}(p)$ representa Francia, ya que París está en Francia, y $\text{pais}(t)$ denota Japón, pues Tokio se encuentra en Japón.

^aNúmero de argumentos que toma una función o predicado

Fórmulas Atómicas

Una vez definidos los términos, podemos construir las **fórmulas atómicas** del lenguaje, las cuales expresan relaciones entre individuos. Estas fórmulas son la base sobre la que se estructuran expresiones lógicas más complejas.

Definición 2. (Fórmulas Atómicas)

Una **fórmula atómica** es una expresión de la forma $P(t_1, \dots, t_n)$ donde P es un **predicado** que establece una relación entre n elementos, y $t_1, \dots, t_n$ son términos que representan individuos del universo de discurso.

Ejemplo 3

Supongamos la siguiente oración:

Luna es una ciudad que tiene playa

En esta expresión, podemos identificar los siguientes elementos:

- ★ **Universo de discurso:** Las ciudades del mundo
- ★ **Predicados:**
 - * $C(x)$: x es una ciudad.
 - * $P(x)$: x tiene playa

Siendo Luna un individuo específico dentro del universo de discurso, podemos representarla con una constante, por ejemplo, l .

Las fórmulas atómicas correspondientes son:

- ★ $C(l)$, que indica que Luna es una ciudad.
- ★ $P(l)$, que expresa que Luna tiene playa.

Fórmulas Compuestas

En el caso de las **fórmulas compuestas**, seguimos utilizando los **conectivos lógicos** que ya conocemos de la Lógica Proposicional para combinar múltiples fórmulas atómicas y expresar relaciones más complejas.

Ejemplo 4

Tomemos nuevamente la oración:

Luna es una ciudad que tiene playa.

La fórmula compuesta correspondiente a este enunciado es:

$$C(l) \wedge P(l)$$

Algunos otros ejemplos de fórmulas compuestas son:

- ★ $M(x) \rightarrow C(x)$ (Si x es una montaña, entonces x es alta).
- ★ $H(x, y) \wedge H(y, x) \rightarrow A(x, z)$ (Si x y y son hermanos, entonces x y z son parientes).

A partir de los ejemplos anteriores, podemos notar que las fórmulas con predicados se construyen de manera similar a las fórmulas de la Lógica Proposicional. La diferencia principal radica en que las fórmulas atómicas ya no son sólo variables proposicionales, sino que ahora representan predicados que incluyen términos, permitiendo una mayor expresividad en la representación del conocimiento.

Fórmulas Cuantificadas

Finalmente, introducimos las **fórmulas con cuantificadores**, las cuales amplían significativamente la expresividad del lenguaje lógico al permitir generalizaciones sobre los individuos del universo de discurso.

Definición 3. (Cuantificación)

Sea φ una fórmula.

- ★ La expresión $\forall x\varphi$ es la **cuantificación universal** de φ con respecto a x y se interpreta como “para todo x , se cumple φ ”.
- ★ La expresión $\exists x\varphi$ es la **cuantificación existencial** de φ con respecto a x y se interpreta como “existe al menos un x tal que se cumple φ ”.

Ejemplo 4

Supongamos que nuestro universo de discurso está formado por los seres vivos. Definimos los siguientes predicados:

- ★ $A(x)$: x es un animal.
- ★ $P(x)$: x es una planta.

A continuación, traducimos algunos enunciados simples que involucren cuantificadores:

- ★ Todo ser vivo es un animal: $\forall xA(x)$
- ★ Todo ser vivo es una planta: $\forall xP(x)$
- ★ Existe al menos un animal y una planta: $\exists x\exists y(A(x) \wedge P(y))$
- ★ Cualquier ser vivo es un animal o una planta: $\forall z(A(z) \vee P(z))$

Variables y Alcance

En lógica de predicados, el **alcance** de una cuantificación determina la región de la fórmula donde una variable es válida.

Definición 4. (Alcance)

Sea $Qx\varphi$ una fórmula de la Lógica de Predicados, donde Q es un cuantificadores ($Q \in \{\forall, \exists\}$), x es una variable y φ es una fórmula bien formada.

El **alcance de la cuantificación** $Qx\varphi$ es la subfórmula φ , es decir, la parte de la expresión donde la variable x se asocia con el cuantificador Qx .

Ejemplo 5

Veamos un ejemplo en el que usamos llaves para indicar el alcance de las variables:

$$\forall x \left(\underbrace{(R(x, m) \wedge R(m, n))}_{x} \rightarrow \exists m \exists n \overbrace{(R(x, m) \wedge R(x, m))}^{m, n} \right)$$

Aquí, la llave más externa muestra el alcance de la cuantificación $\forall x$, mientras que la llave interna indica el alcance de los cuantificadores $\exists m \exists n$.

Es importante notar que un cuantificador sólo afecta la fórmula que aparece inmediatamente a su derecha. Por ejemplo, en la expresión:

$$\forall x S(a) \wedge T(x)$$

la variable x en $T(x)$ no está dentro del alcance de $\forall x$. Para incluirla, debemos agrupar la expresión correctamente con paréntesis:

$$\forall x (S(a) \wedge T(x))$$

Tipos de Variables

Distinguimos tres tipos de variables en lógica de predicados:

Variables de Ligado

Definición 5. (Variable de Ligado)

Una **variable de ligado** es aquella que aparece junto a un cuantificador, indicando que debe interpretarse dentro de un alcance específico.

Ejemplo 6

En las fórmulas $\forall xQ(x)$ y $\exists zM(z)$, las variables de ligado son x y z , pues aparecen junto a los cuantificadores.

Variables Ligadas

Definición 6. (Variable Ligada)

Una **variable está ligada** si aparece dentro del alcance de un cuantificador que la declara explícitamente. En Lógica de Predicados, todas las variables que coinciden en nombre con la variable de ligado de un cuantificador se consideran ligadas dentro de su alcance.

Ejemplo 7

En la fórmula $\forall xQ(x)$, la variable x en $Q(x)$ está ligada. Sin embargo, en la expresión $\forall yT(w)$, no hay ninguna variable ligada, ya que w no aparece cuantificada.

Variables Libres

Definición 7. (Variable Libre)

Una **variable está libre** si no se encuentra dentro del alcance de un cuantificador que la ligue. En Lógica de Predicados, todas las variables que no son ligadas por un cuantificador en una fórmula son libres.

Ejemplo 8

En la fórmula $\forall xQ(x)$, no hay variables libres, pues x está ligadas. Sin embargo, en la expresión $\forall yT(w)$, la variable w es libre, ya no es cuantificada.

Ejercicio 1

Para cada una de las siguientes fórmulas, indica el alcance de cada cuantificador y señala las variables de ligado, ligadas y libres.

1. $\forall x(P(x) \vee \exists yQ(y, z))$
2. $\exists a(R(a, b) \wedge \forall cS(a, c))$
3. $\forall x\forall y(F(x, y) \rightarrow \exists zG(y, z))$
4. $\forall u(M(u) \vee N(v)) \rightarrow \exists wH(u, w)$
5. $\forall x(A(x) \wedge \exists y(B(y) \vee C(x, y))) \vee D(z)$

2. Formalización del Lenguaje

Ahora que hemos descrito de manera informal al lenguaje de la Lógica de Predicados (LPred) y sus componentes fundamentales, podemos representarlo de manera más formal y estructurada mediante una gramática en notación EBNF. Siguiendo un enfoque similar al utilizado para la Lógica Proposicional (LProp), definiremos las reglas de formación del lenguaje en términos de producciones sintácticas, lo que permitirá describir formalmente como se construyen expresiones bien formadas en LPred.

$$\begin{aligned}
 \langle \text{cte} \rangle & ::= \perp \mid \top \\
 \langle \text{var} \rangle & ::= x \mid y \mid z \mid \dots \\
 \langle \text{func} \rangle & ::= f \mid g \mid \dots \\
 \langle \text{term} \rangle & ::= \langle \text{var} \rangle \\
 & \quad \mid c \quad (\text{donde } c \text{ es una constante}) \\
 & \quad \mid \langle \text{func} \rangle (\langle \text{term} \rangle, \dots, \langle \text{term} \rangle) \\
 \langle \text{pred} \rangle & ::= P \mid Q \mid \dots \\
 \langle \text{atom} \rangle & ::= \langle \text{cte} \rangle \\
 & \quad \mid \langle \text{pred} \rangle (\langle \text{term} \rangle, \dots, \langle \text{term} \rangle) \\
 \langle \text{binop} \rangle & ::= \vee \mid \wedge \mid \rightarrow \mid \leftrightarrow \\
 \langle \text{quant} \rangle & ::= \forall \mid \exists \\
 \langle \text{expr} \rangle & ::= \langle \text{atom} \rangle \\
 & \quad \mid (\neg \langle \text{expr} \rangle) \\
 & \quad \mid (\langle \text{expr} \rangle \langle \text{binop} \rangle \langle \text{expr} \rangle) \\
 & \quad \mid (\langle \text{quant} \rangle \langle \text{var} \rangle \langle \text{expr} \rangle)
 \end{aligned}$$

Es importante notar que el lenguaje LPred no está definido de manera única, sino que se compone de dos partes fundamentales: una **parte común**, presente en todos los lenguajes de primer orden, y una **parte particular**, que varía según la especificación del lenguaje.

Parte Común

La parte común está formada por los siguientes elementos, que son compartidos por todos los lenguajes de primer orden:

1. Constantes lógicas ($\perp, \top$).
2. Conectivos lógicos ($\vee, \wedge, \rightarrow, \leftrightarrow, \neg$).
3. Cuantificadores ($\forall, \exists$).
4. Símbolos auxiliares (paréntesis, comas).

Parte Particular

Lo que diferencia a un lenguaje de otro es su **signatura**, la cual define los símbolos específicos que se utilizarán. Una signatura $\mathcal{L}$, se compone de:

1. Un conjunto $\mathcal{P}$ de **símbolos de predicado**, que determinan las relaciones entre los términos.
2. Un conjunto $\mathcal{F}$ de **símbolos de función**, que permiten construir términos más complejos.

3. Un conjunto $\mathcal{C}$ de **símbolos de constante**, que representan individuos específicos dentro del universo de discurso.

De este modo, un lenguaje de primer orden queda completamente determinado por su signatura, especificando los símbolos que lo distinguen de otros lenguajes.

$$\mathcal{L} = \mathcal{P} \cup \mathcal{F} \cup \mathcal{C}$$

A continuación, analizamos las definiciones que se generar a partir de la gramática para representar formalmente a los términos y fórmulas del lenguaje.

Términos ($\mathbf{TERM}_{\mathcal{L}}$)

$$\begin{aligned} \langle \text{term} \rangle & ::= \langle \text{var} \rangle \\ & | c \quad (\text{donde } c \text{ es una constante}) \\ & | \langle \text{func} \rangle(\langle \text{term} \rangle, \dots, \langle \text{term} \rangle) \end{aligned}$$

Los términos de un lenguaje de primer orden bajo una signatura $\mathcal{L}$ se definen recursivamente de la siguiente manera:

1. Las constantes son términos.
2. Las variables son términos.
3. Si f es un símbolo de función de aridad n y $t_1, \dots, t_n$ son términos, entonces la expresión $f(t_1, \dots, t_n)$ es un término.
4. Son todas.¹

Fórmulas Atómicas ($\mathbf{ATOM}_{\mathcal{L}}$)

$$\begin{aligned} \langle \text{atom} \rangle & ::= \langle \text{cte} \rangle \\ & | \langle \text{pred} \rangle(\langle \text{term} \rangle, \dots, \langle \text{term} \rangle) \end{aligned}$$

Siguiendo el mismo enfoque que en la Lógica Proposicional, comenzamos definiendo las fórmulas atómicas en un lenguaje de primer orden. Estas se componen de:

1. Las constantes lógicas: $\perp, \top$.
2. Las expresiones de la forma $P(t_1, \dots, t_n)$, donde $t_1, \dots, t_n \in \mathbf{TERM}_{\mathcal{L}}$ y P es un símbolo de predicado.

¹Recordemos que en una definición recursiva, es obligatorio incluir esta cláusula para asegurar que no existen otros objetos en la definición. De lo contrario, podrían introducirse elementos no contemplados, afectando la precisión y el cierre de la definición.

Fórmulas Compuestas ($\text{LPred}_{\mathcal{L}}$)

$$\begin{aligned} \langle \text{expr} \rangle & ::= \langle \text{atom} \rangle \\ & | (\neg \langle \text{expr} \rangle) \\ & | (\langle \text{expr} \rangle \langle \text{binop} \rangle \langle \text{expr} \rangle) \\ & | (\langle \text{quant} \rangle \langle \text{var} \rangle \langle \text{expr} \rangle) \end{aligned}$$

El conjunto $\text{LPred}_{\mathcal{L}}$ de expresiones compuestas aceptadas en una signatura $\mathcal{L}$, llamadas simplemente fórmulas, se define de manera recursiva de la siguiente forma:

1. Si $\varphi \in \text{ATOM}_{\mathcal{L}}$, entonces $\varphi \in \text{LPred}_{\mathcal{L}}$.
2. Si $\varphi \in \text{LPred}_{\mathcal{L}}$, entonces $(\neg\varphi) \in \text{LPred}_{\mathcal{L}}$.
3. Si $\varphi, \psi \in \text{LPred}_{\mathcal{L}}$, entonces $(\varphi \star \psi) \in \text{LPred}_{\mathcal{L}}$, donde $\star \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$.
4. Si $\varphi \in \text{LPred}_{\mathcal{L}}$ y x es una variable, entonces $(\forall x\varphi), (\exists x\varphi) \in \text{LPred}_{\mathcal{L}}$.
5. Son todas.

Cada lenguaje definido a partir de una signatura y conforme a estas reglas es un lenguaje de primer orden.

Observación 1

Es importante tener en cuenta las siguientes consideraciones fundamentales en la Lógica de Predicados:

1. Los términos y las fórmulas son categorías sintácticas distintas: ningún término es una fórmula y ninguna fórmula es un término.
2. Los términos representan exclusivamente individuos u objetos dentro del universo de discurso.
3. Las fórmulas atómicas expresan propiedades sobre los términos, estableciendo relaciones entre ellos.
4. Sólo los individuos u objetos pueden ser cuantificados y sobre ellos se pueden formular predicados. Esta característica es la que da origen al nombre **Lógica de Primer Orden**, ya que los cuantificadores no pueden aplicarse sobre predicados o funciones, sino únicamente sobre los elementos del universo de discurso.

Ejercicio 2

Determina si las siguientes expresiones pertenecen a LPred y justifica la respuesta usando la gramática EBNF:

1. $P(a) \vee \forall xQ(x)$
2. $\forall x(R(x, y) \wedge S(y, z))$
3. $\exists x(f(x) \rightarrow P(x))$
4. $\forall P(P(a) \vee Q(a))$
5. $\neg \forall x \exists y (M(x, y) \wedge \rightarrow N(y))$

3. Inducción y Recursión

Las definiciones recursivas de los elementos del lenguaje establece naturalmente principios de inducción. Dado que en la Lógica de Predicados existen dos categorías sintácticas (términos y fórmulas) también contamos con principios de inducción específicos para cada una de ellas.

Definición 8. Principio de Inducción Estructural para $TERM_{\mathcal{L}}$

Sea P una propiedad definida sobre términos. Para demostrar que P es válida para todos los términos del lenguaje, se sigue el procedimiento:

Caso base

1. Si x es una variable, demostrar que $P(x)$.
2. Si c es una constante, demostrar que $P(c)$.

Hipótesis de inducción

Suponer que la propiedad $\mathcal{P}$ se cumple para cualesquiera términos $t_1, \dots, t_n \in TERM_{\mathcal{L}}$, es decir:

$$\mathcal{P}(t_1), \dots, \mathcal{P}(t_n)$$

Paso inductivo

Si f es un símbolo de función de aridad n , demostrar que la propiedad también se cumple para $f(t_1, \dots, t_n)$ es decir:

$$\mathcal{P}(f(t_1, \dots, t_n))$$

Definición 9. Principio de Inducción Estructural para $LPred_{\mathcal{L}}$

Sea $\mathcal{P}$ una propiedad sobre fórmulas. Para demostrar que $\mathcal{P}$ se cumple para todas las fórmulas del lenguaje, se procede de la siguiente manera:

Caso base

Mostrar que toda fórmula atómica satisface $\mathcal{P}$.

Hipótesis de inducción

Suponer que la propiedad $\mathcal{P}$ es válida para cualesquiera fórmulas $\varphi, \psi \in LPred_{\mathcal{L}}$, es decir:

$$\mathcal{P}(\varphi) \quad \mathcal{P}(\psi)$$

Paso inductivo

1. Demostrar que $\mathcal{P}(\neg\varphi)$.
2. Demostrar que $\mathcal{P}(\varphi \star \psi)$, donde $\star \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$.
3. Demostrar que $\mathcal{P}(\forall x\varphi)$ y $\mathcal{P}(\exists x\varphi)$.

Ejemplo 9

```

preds :: LPred -> Int
preds ⊥ = 0
preds ⊤ = 0
preds P(x1, ..., xn) = 1
preds (¬φ) = preds φ
preds (φ ★ ψ) = preds φ + preds ψ
preds (∀xφ) = preds φ
preds (∃xφ) = preds φ

subf :: LPred -> Int
subf ⊥ = 1
subf ⊤ = 1
subf P(x1, ..., xn) = 1
subf (¬φ) = 1 + subf φ
subf (φ ★ ψ) = 1 + subf φ + subf ψ
subf (∀xφ) = 1 + subf φ
subf (∃xφ) = 1 + subf φ

```

Proposición

En toda fórmula $\varphi \in \text{LPred}_{\mathcal{L}}$, se cumple:

$$\text{preds}(\varphi) \leq \text{subf}(\varphi)$$

Demostración por inducción estructural**Caso base:**

- * $\varphi = \perp$ o $\varphi = \top$:
 $\text{preds}(\varphi) = 0, \text{subf}(\varphi) = 1 \Rightarrow 0 \leq 1$
- * $\varphi = P(t_1, \dots, t_n)$:
 $\text{preds}(\varphi) = 1, \text{subf}(\varphi) = 1 \Rightarrow 1 \leq 1$

Paso inductivo:

Supongamos que la propiedad se cumple para ψ y θ .

- * $\varphi = \neg\psi$:
 $\text{preds}(\varphi) = \text{preds}(\psi)$
 $\text{subf}(\varphi) = 1 + \text{subf}(\psi)$
 Por hipótesis inductiva, $\text{preds}(\psi) \leq \text{subf}(\psi) \Rightarrow \text{preds}(\varphi) \leq \text{subf}(\varphi)$
- * $\varphi = \psi \star \theta$:
 $\text{preds}(\varphi) = \text{preds}(\psi) + \text{preds}(\theta)$
 $\text{subf}(\varphi) = 1 + \text{subf}(\psi) + \text{subf}(\theta)$
 Por hipótesis: $\text{preds}(\psi) \leq \text{subf}(\psi), \text{preds}(\theta) \leq \text{subf}(\theta)$
 Sumando: $\text{preds}(\varphi) \leq \text{subf}(\varphi)$
- * $\varphi = Qx\psi$:
 $\text{preds}(\varphi) = \text{preds}(\psi)$
 $\text{subf}(\varphi) = 1 + \text{subf}(\psi)$
 Por hipótesis: $\text{preds}(\psi) \leq \text{subf}(\psi) \Rightarrow \text{preds}(\varphi) \leq \text{subf}(\varphi)$

Conclusión:

Por inducción estructural sobre $\varphi \in \text{LPred}_{\mathcal{L}}$, se cumple:

$$\text{preds}(\varphi) \leq \text{subf}(\varphi) \quad \blacksquare$$

Sustitución

La noción de **sustitución** en la lógica de predicados es más compleja que en la lógica proposicional, debido a la presencia de términos y variables ligadas. A diferencia de la lógica proposicional, donde la sustitución consiste únicamente en una operación textual sobre las fórmulas, en la lógica de predicados las sustituciones se aplican tanto a términos como a fórmulas. En este último caso, deben respetar el ligado de las variables, por lo que no pueden considerarse operaciones textuales.

Definición 10. (Sustitución en Lógica de Predicados)

Una **sustitución** en un lenguaje de predicados $\mathcal{L}$ es una expresión del tipo

$$[x_1, \dots, x_n := t_1, \dots, t_n]$$

donde se cumple que:

- ★ $x_1, \dots, x_n$ son **variables distintas** del lenguaje $\mathcal{L}$,
- ★ $t_1, \dots, t_n$ son **términos** de $\mathcal{L}$,
- ★ Para todo $1 \leq i \leq n$, se tiene que $x_i \neq t_i$ (es decir, **una variable no se sustituye por sí misma**).

A esta sustitución también se le puede denotar de forma compacta como $[\vec{x} := \vec{t}]$.

Esta definición formaliza la idea de reemplazar ciertas variables por términos en una fórmula o término del lenguaje. La sustitución $[x_1, \dots, x_n := t_1, \dots, t_n]$ nos indica que, en cualquier expresión donde aparezca una de las variables x_i , debe reemplazarse por el correspondiente término t_i . Es importante que las variables sean distintas para evitar ambigüedades, y que ninguna variable se sustituya por sí misma, ya que esto no aportaría ningún cambio. Esta operación es fundamental en la lógica de primer orden porque nos permite manipular expresiones con variables de manera controlada, especialmente cuando trabajamos con cuantificadores y reglas de inferencia.

Sustitución en Términos

La **aplicación de una sustitución** $[\vec{x} := \vec{t}]$ a un término r , denotada por $r[\vec{x} := \vec{t}]$, consiste en reemplazar **simultáneamente** cada ocurrencia de la variable x_i en r por el correspondiente término t_i . Este procedimiento se define **recursivamente** de la siguiente manera:

$$\begin{aligned} x_i[\vec{x} := \vec{t}] &= t_i & 1 \leq i \leq n \\ z[\vec{x} := \vec{t}] &= z & z \neq x_i, 1 \leq i \leq n \\ c[\vec{x} := \vec{t}] &= c & c \in \mathcal{C} \\ f(t_1, \dots, t_m)[\vec{x} := \vec{t}] &= f(t_1[\vec{x} := \vec{t}], \dots, t_m[\vec{x} := \vec{t}]) & f^{(m)} \in \mathcal{F} \end{aligned}$$

Esta definición nos dice cómo realizar una sustitución paso a paso según la forma del término. Si el término es una variable que está en la lista de sustitución, se reemplaza por el término correspondiente. Si es una variable distinta, o una constante, se deja igual. Y si el término es una aplicación de una función, se aplica la sustitución recursivamente a cada uno de sus argumentos. Este enfoque recursivo permite extender la sustitución a términos complejos contruidos a partir de funciones y otros términos.

Sustitución en Fórmulas

La aplicación de una sustitución $[\vec{x} := \vec{t}]$ a una fórmula φ , denotada por $\varphi[\vec{x} := \vec{t}]$, se define recursivamente de la siguiente manera:

$$\begin{aligned} \perp[\vec{x} := \vec{t}] &= \perp \\ \top[\vec{x} := \vec{t}] &= \top \\ P(t_1, \dots, t_m)[\vec{x} := \vec{t}] &= P(t_1[\vec{x} := \vec{t}], \dots, t_m[\vec{x} := \vec{t}]) \\ (\neg\varphi)[\vec{x} := \vec{t}] &= \neg(\varphi[\vec{x} := \vec{t}]) \\ (\varphi \star \psi)[\vec{x} := \vec{t}] &= (\varphi[\vec{x} := \vec{t}] \star \psi[\vec{x} := \vec{t}]) \quad \star \in \{\wedge, \vee, \rightarrow, \leftrightarrow\} \end{aligned}$$

Esta definición describe cómo aplicar una sustitución a una fórmula lógica siguiendo su estructura. Para los símbolos de verdad $\perp$ y $\top$, no hay cambio. En el caso de predicados, la sustitución se aplica a cada uno de los términos que aparecen como argumentos. Cuando la fórmula está construida con conectivos lógicos como la negación, disyunción, conjunción, condicional o bicondicional, la sustitución se aplica recursivamente a cada subfórmula.

Este proceso asegura que las variables designadas en la sustitución sean reemplazadas correctamente, respetando la estructura lógica de la fórmula.

La aplicación de sustituciones a fórmulas que contienen cuantificadores requiere especial atención, ya que estas fórmulas involucran variables ligadas. Si se aplica una sustitución de manera puramente textual, pueden surgir problemas tanto sintácticos como semánticos.

Por ejemplo, al aplicar la sustitución $[x := c]$ a la fórmula $\forall xP(x, x)$, se obtiene directamente:

$$\forall xP(x, x)[x := c]$$

Si interpretamos esta sustitución de forma textual, podríamos llegar erróneamente a $\forall xP(c, c)$, lo cual altera el significado original de la fórmula. En efecto, la variable x está ligada por el cuantificador, por lo que no debería ser sustituida. Otro caso se presenta cuando tenemos una fórmula como:

$$\forall x\exists yP(y, x)[y := x]$$

La fórmula original expresa que, para cada elemento x del universo de discurso, existe un elemento y tal que se cumple $P(y, x)$. Sin embargo, al aplicar la sustitución obtenemos $P(x, x)$, lo cual altera completamente el significado de la fórmula. Esto sucede porque la sustitución provoca que dos variables que antes eran independientes queden ligadas.

Estos ejemplos ilustran que la sustitución en fórmulas no puede realizarse de manera meramente textual, ya que puede conducir a resultados que no preservan el significado lógico original.

Tenemos dos formas de evitar estos problemas:

Evitar la sustitución

$$\begin{aligned}
 (\forall x\varphi) [\vec{x} := \vec{t}] &= (\forall x\varphi) && x \in \vec{x} \cup \text{Var}(\vec{t}) \\
 (\forall x\varphi) [\vec{x} := \vec{t}] &= \forall x (\varphi[\vec{x} := \vec{t}]) && x \notin \vec{x} \cup \text{Var}(\vec{t}) \\
 (\exists x\varphi) [\vec{x} := \vec{t}] &= (\exists x\varphi) && x \in \vec{x} \cup \text{Var}(\vec{t}) \\
 (\exists x\varphi) [\vec{x} := \vec{t}] &= \exists x (\varphi[\vec{x} := \vec{t}]) && x \notin \vec{x} \cup \text{Var}(\vec{t})
 \end{aligned}$$

Ejemplo 10

Realizar la siguiente sustitución:

$$\exists x Q(y, z, y)[y, z := g(d), f(y)]$$

Solución:

1. Como $x \notin \vec{x}$ ni $x \in \text{Var}(\vec{t})$, podemos sustituir bajo el cuantificador.
2. Aplicamos la sustitución:

$$\exists x (Q(y, z, y)[y, z := g(d), f(y)])$$

3. Sustituimos término a término:

$$\exists x Q(g(d), f(y), g(d))$$

Ejemplo 11

Realizar la siguiente sustitución:

$$\exists x (Q(y, z, y) \wedge \forall w P(w))[y, z := g(d), f(y)]$$

Solución:

1. Como $x \notin \vec{x}$ ni $x \in \text{Var}(\vec{t})$, podemos sustituir bajo el cuantificador.
2. Aplicamos la sustitución:

$$\exists x ((Q(y, z, y) \wedge \forall w P(w))[y, z := g(d), f(y)])$$

3. Distribuimos la sustitución:

$$\exists x (Q(y, z, y)[y, z := g(d), f(y)] \wedge \forall w P(w)[y, z := g(d), f(y)])$$

4. Aplicamos sustituciones:

$$\exists x (Q(g(d), f(y), g(d)) \wedge \forall w P(w))$$

Observación 2

Este método falla en casos como:

$$\forall x (P(x, x) \rightarrow P(y, y))[x, y := a, b]$$

pues no permite sustituir correctamente a y . Necesitamos una estrategia más robusta.

Forma 2: Usar α -equivalencias

Podemos cambiar el nombre de las variables ligadas sin alterar el significado de una fórmula. Por ejemplo:

$$\forall xP(x) \quad \text{y} \quad \forall yP(y)$$

expresan exactamente el mismo significado: da igual el nombre que utilicemos para las variables ligadas, el resultado lógico de la fórmula es el mismo.

Definición 11. (α -equivalencia)

Dos fórmulas φ_1 y φ_2 son α -equivalentes (se denota $\varphi_1 \sim_\alpha \varphi_2$) si una puede obtenerse de la otra mediante un cambio de nombres de sus variables ligadas, sin modificar ni las variables libres ni la estructura lógica de la fórmula.

Cuando hablamos de α -equivalencia, nos referimos a que el nombre de las variables ligadas no importa: podemos renombrarlas libremente mientras respetemos su función y relación dentro de la fórmula. Es como cambiar el nombre de un personaje en una historia: si el rol y las relaciones se mantienen, la historia sigue siendo la misma.

Con α -equivalencias, la definición de sustitución se vuelve total:

$$\begin{aligned} (\forall x\varphi)[\vec{x} := \vec{t}] &= (\forall y\psi)[\vec{x} := \vec{t}] && \text{si } x \in \vec{x} \cup \text{Var}(\vec{t}) \text{ y } (\forall x\varphi) \sim_\alpha (\forall y\psi) \\ (\forall x\varphi)[\vec{x} := \vec{t}] &= \forall x(\varphi[\vec{x} := \vec{t}]) && \text{si } x \notin \vec{x} \cup \text{Var}(\vec{t}) \\ (\exists x\varphi)[\vec{x} := \vec{t}] &= (\exists y\psi)[\vec{x} := \vec{t}] && \text{si } x \in \vec{x} \cup \text{Var}(\vec{t}) \text{ y } (\exists x\varphi) \sim_\alpha (\exists y\psi) \\ (\exists x\varphi)[\vec{x} := \vec{t}] &= \exists x(\varphi[\vec{x} := \vec{t}]) && \text{si } x \notin \vec{x} \cup \text{Var}(\vec{t}) \end{aligned}$$

Ejemplo 12

Realizar la siguiente sustitución:

$$\exists zQ(y, z, y)[y, z := g(d), f(y)]$$

Solución:

1. Aplicamos una α -equivalencia para evitar conflicto:

$$\exists zQ(y, z, y) \sim_\alpha \exists wQ(y, w, y)$$

2. Aplicamos la sustitución:

$$\exists w(Q(y, w, y)[y, z := g(d), f(y)])$$

3. Realizamos las sustituciones correspondientes:

$$\exists wQ(g(d), w, g(d))$$

4. Automatización

En las secciones anteriores vimos que la construcción de términos y fórmulas en la Lógica de Predicados sigue un esquema inductivo que puede modelarse fácilmente en un lenguaje de programación como HASKELL.

Esta estructura inductiva abre la posibilidad de **automatizar** la verificación y manipulación de fórmulas, permitiendo:

- ★ Verificar si una expresión es una fórmula bien formada.
- ★ Aplicar sustituciones respetando el alcance de las variables.
- ★ Detectar variables libres y ligadas.
- ★ Preparar fórmulas para procesos como unificación o resolución binaria que estudiaremos más adelante.

Modelado de la sintaxis

Tal como se ha hecho en el modelado previo de términos y fórmulas, la sintaxis de la lógica de predicados puede representarse mediante tipos de datos algebraicos. Podemos definir los tipos:

```
data Term = Var Simbolo | Fun Simbolo [Term] deriving (Eq, Show)

data Formula = Top
             | Bottom
             | Predicado Simbolo [Term]
             | Not Formula
             | And Formula Formula
             | Or Formula Formula
             | Impl Formula Formula
             | Iff Formula Formula
             | ForAll Simbolo Formula
             | Exists Simbolo Formula deriving (Eq, Show)
```

Con esta representación, las reglas de construcción del lenguaje (definidas por su gramática) quedan reflejadas en los constructores de datos.

Funciones básicas de automatización

A partir de estos tipos, es posible definir funciones que procesen las fórmulas sintácticamente de manera automática. Por ejemplo:

- ★ **Verificación de fórmulas** Funciones que validan si una estructura construida cumple la sintaxis de LPred.
- ★ **Conteo de predicados o subfórmulas** Usando recursión.
- ★ **Detección de variables libres** Funciones que recorren la fórmula buscando variables no ligadas por cuantificadores.

- ★ **Aplicación de sustituciones** De acuerdo con las reglas definidas formalmente para términos y fórmulas.

Por ejemplo, una función que recolecta todas las variables libres de una fórmula podría ser:

```
varsLibres :: Formula -> [Simbolo]
varsLibres Top = []
varsLibres Bottom = []
varsLibres (Predicado _ ts) = concatMap varsTermino ts
varsLibres (Not f) = varsLibres f
varsLibres (And f g) = varsLibres f ++ varsLibres g
varsLibres (Or f g) = varsLibres f ++ varsLibres g
varsLibres (Impl f g) = varsLibres f ++ varsLibres g
varsLibres (Iff f g) = varsLibres f ++ varsLibres g
varsLibres (ForAll x f) = filter (/= x) (varsLibres f)
varsLibres (Exists x f) = filter (/= x) (varsLibres f)

varsTermino :: Term -> [Simbolo]
varsTermino (Var x) = [x]
varsTermino (Fun _ ts) = concatMap varsTermino ts
```

Esta función permite detectar automáticamente qué variables requieren ser cuantificadas para convertir una fórmula en una **fórmula cerrada**.

Observación 3

La automatización de la sintaxis de la Lógica de Primer Orden nos permite pasar de un enfoque manual y propenso a errores a un enfoque programático, confiable y escalable, preparando el terreno para el desarrollo de sistemas de razonamiento automático más complejos.

5. Conclusión

La Lógica de Predicados extiende la capacidad expresiva de la Lógica Proposicional, permitiendo representar y razonar sobre propiedades y relaciones entre individuos de un universo de discurso. A través del estudio de su sintaxis, hemos formalizado la construcción de términos, fórmulas atómicas y compuestas, así como el uso de cuantificadores para expresar generalizaciones y existencia. La formalización mediante gramáticas EBNF, junto con los principios de inducción estructural, nos brinda las herramientas necesarias para definir de manera precisa y rigurosa el lenguaje lógico.

Además, la automatización de estas estructuras, ejemplificada mediante la programación funcional, muestra cómo la teoría puede convertirse en práctica efectiva. Modelar términos y fórmulas como tipos de datos algebraicos facilita no sólo la validación sintáctica de expresiones, sino también su procesamiento en tareas más avanzadas como la verificación formal, la unificación y la resolución. De esta forma, el dominio de la sintaxis no sólo representa un paso fundamental en la formación lógica, sino que también constituye la base para construir sistemas inteligentes de razonamiento automático.

Referencias

- [1] Miranda Perea, F. E., Reyes, A. L., González, L. del C., & Linares, S. (2018). *Lógica Computacional: Notas de clase*. Facultad de Ciencias, Universidad Nacional Autónoma de México.

- [2] Loyola Cruz, L. F. (2023). *Manual de Prácticas para la Asignatura de Lógica Computacional* (Proyecto de Apoyo a la Docencia). Universidad Nacional Autónoma de México.
- [3] Enderton, H. B. (2001). *A Mathematical Introduction to Logic* (2nd ed.). Elsevier.
- [4] Huth, M., & Ryan, M. (2004). *Logic in Computer Science: Modelling and Reasoning about Systems* (2nd ed.). Cambridge University Press.
- [5] Mendelson, E. (2015). *Introduction to Mathematical Logic* (6th ed.). CRC Press.
- [6] Harrison, J. (2009). *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press.