Lógica Computacional, 2025-2

Unidad 3: Lógica de Predicados

Resolución Binaria con Unificación

Manuel Soto Romero

Luis Fernando Loyola Cruz

18 de mayo de 2025
Facultad de Ciencias UNAM

En esta nota profundizaremos en uno de los métodos más potentes para la inferencia automática en lógica de predicados: la resolución binaria con unificación. Partiremos de lo ya aprendido en lógica proposicional, donde utilizamos la resolución como técnica de decisión, y extendaremos ese enfoque al ámbito más expresivo de la lógica de predicados. Para ello, desarrollaremos un conjunto de herramientas fundamentales: aprenderemos a transformar fórmulas a su forma clausular mediante una serie de pasos sistemáticos (rectificación, forma normal negativa, forma prenex y skolemización) y estudiaremos detalladamente el proceso de unificación, indispensable para aplicar resolución en este nuevo contexto.

Además, mostraremos cómo estos procedimientos pueden ser formalizados y ejecutados de forma automática utilizando el lenguaje de programación funcional HASKELL. A través de funciones puras que modelan fórmulas, sustituciones y cláusulas, veremos cómo implementar verificadores lógicos y simuladores del algoritmo de resolución. Este enfoque computacional no solo refuerza la comprensión teórica del proceso, sino que ofrece una introducción práctica a la automatización del razonamiento lógico y a los fundamentos de la deducción automática.

1. Introducción

Al igual que en la lógica proposicional, podemos apoyarnos de la regla de resolución binaria de Robinson como un método sintáctico para decidir la corrección de un argumento.

La idea es la misma que en la lógica proposicional; es decir, dado un conjunto de cláusulas, se busca llegar a la cláusula vacía para mostrar que un argumento es correcto o que una fórmula es válida. Sin embargo, es necesario hacer un par de adecuaciones.

1. Para encontrar la forma clausular de una fórmula, no nos será suficiente el método que estudiamos en lógica proposicional, debido a la presencia de cuantificadores. Por ello, necesitamos establecer un nuevo proceso que nos permita obtener esta forma utilizando nuevas formas normales.
2. Por otro lado, recordemos que en la lógica proposicional, la cláusula vacía se obtiene al resolver dos literales contrarias, por ejemplo $p, \neg p$. En la lógica de primer orden, las literales son fórmulas atómicas, es decir, predicados. Sin embargo, a diferencia de la lógica proposicional, resolver dos literales en lógica de primer orden requiere mayor cuidado. Por ejemplo $P(a, x), \neg P(z, b)$ no puede resolverse, aún cuando comparten el mismo símbolo de predicado, pues no son estrictamente contrarias: no coinciden en las variables y/o constantes ni en su disposición.

Sin embargo, en ciertos casos (como veremos en breve) es posible aplicar una sustitución que **unifique** literales y permita aplicar la regla de resolución. En el ejemplo anterior, podemos utilizar la sustitución $\sigma = [x, z := b, a]$, con la cual se obtienen las literales $P(a, b)$ y $\neg P(a, b)$, que ahora sí son contrarias.

Por lo tanto, en las siguientes secciones nos centraremos en el estudio de la obtención de formas normales, así como en un método para la búsqueda y aplicación de sustituciones que permitan obtener literales contrarias, conocido como **unificación**.

2. Formas Normales

El objetivo es obtener una forma clausular que nos permita aplicar el método de resolución binaria en lógica de primer orden. El proceso general que seguimos consta de los siguientes pasos:

1. Rectificación de fórmulas
2. Transformación a forma normal negativa (FNN)
3. Transformación a forma normal prenex (FNP)
4. Eliminación de cuantificadores mediante la forma normal de Skolem (FNS)
5. Conversión a forma clausular

Se obtiene así una fórmula de la forma:

$$\forall x_1 \dots x_n (\mathcal{C}_1 \wedge \dots \wedge \mathcal{C}_k)$$

donde $\mathcal{C}_1 \wedge \dots \wedge \mathcal{C}_k$ representa una conjunción de clausulas. Dado que todos los cuantificadores son universales y abarcan toda la fórmula, es común omitirlos por simplicidad.

Rectificación

La **rectificación** garantiza que cada cuantificador utilice variables con nombres distintos y que ninguna variable ligada aparezca como libre. Además, permite eliminar cuantificadores vacíos o redundantes.

Definición 1. (Rectificación de una fórmula)

Una fórmula φ está **rectificada** si y sólo si cumple las siguientes condiciones:

1. Ninguna variable aparece en φ al mismo tiempo como **libre** y como **ligada** por un cuantificador.
2. No hay dos cuantificadores distintos que usen el **mismo nombre de variable** en subfórmulas con **alcances independientes**.
3. No existen cuantificadores múltiples **consecutivos** que usen el mismo nombre de variable, como en las expresiones:

$$\forall x \forall x \varphi \quad \exists x \exists x \varphi \quad \forall x \exists x \varphi$$

4. No hay **cuantificadores vacíos**, es decir, cada variable introducida por un cuantificador aparece al menos una vez dentro de su alcance.

La rectificación garantiza que las variables utilizadas en una fórmula estén correctamente organizadas y sin ambigüedades, evitando situaciones como que una misma variable sea libre y ligada a la vez, que se repitan

nombres en distintos cuantificadores o que se introduzcan variables innecesarias. Esto es comparable al análisis estático que realizan los compiladores, donde se revisa que el uso de las variables en un programa sea coherente antes de compilar. Así como en programación se evitan variables sin uso, duplicadas o con nombres ambiguos, en lógica formal la rectificación es un paso fundamental para preparar la fórmula antes de aplicar transformaciones automáticas como la skolemización o la resolución.

Algunas equivalencias útiles:

- ★ $\forall x \forall x \varphi \equiv \forall x \varphi$
- ★ $\exists x \exists x \varphi \equiv \exists x \varphi$
- ★ $\exists x \forall x \varphi \equiv \forall x \varphi$
- ★ $\forall x \exists x \varphi \equiv \exists x \varphi$
- ★ $\forall x \varphi \equiv \forall y (\varphi[x := y])$ si y no aparece libre en φ
- ★ $\exists x \varphi \equiv \exists y (\varphi[x := y])$ si y no aparece libre en φ
- ★ $\forall x \varphi \equiv \varphi$ si x no aparece libre en φ
- ★ $\exists x \varphi \equiv \varphi$ si x no aparece libre en φ

Ejemplo 1

Sea la fórmula:

$$\varphi = \forall x \exists y \neg \forall w \exists z (P(x, y) \vee \neg Q(x) \rightarrow \exists w \neg T(a, w))$$

Encontrar una versión *rectificada* de φ ; es decir, una fórmula lógicamente equivalente en la que no haya colisión de nombres de variables, ni cuantificadores redundantes, vacíos o ambiguos.

Paso 1: Identificar conflictos de nombres

Observamos que la variable w aparece:

- ★ como variable ligada por el cuantificador $\forall w$, con alcance sobre toda la subfórmula;
- ★ y nuevamente ligada por un cuantificador $\exists w$, en una subfórmula más interna.

Esto viola la condición de *no cuantificar una misma variable en distintos alcances anidados*, por lo tanto, debemos renombrar la segunda aparición de w .

Paso 2: Sustituir nombres repetidos

Renombramos la segunda instancia de w por una variable nueva, digamos u , que no aparezca previamente en la fórmula. Entonces, la versión rectificada es:

$$\varphi \equiv \forall x \exists y \neg \forall w \exists z (P(x, y) \vee \neg Q(x) \rightarrow \exists u \neg T(a, u))$$

Rectificar una fórmula consiste, en este caso, en evitar la reutilización de un nombre de variable cuantificada. Aunque en términos lógicos se puede interpretar, esta práctica puede causar ambigüedad en procesos automáticos como la skolemización o la aplicación de sustituciones. Al reemplazar el nombre repetido por una variable nueva (como $w \rightarrow u$), mantenemos la equivalencia lógica y garantizamos la corrección sintáctica.

Forma normal negativa

La Forma Normal Negativa, como vimos en lógica proposicional, tiene como objetivo empujar las negaciones hasta las fórmulas atómicas, además de eliminar los condicionales y bicondicionales presentes en la fórmula.

Definición 2. (Forma normal negativa)

Una fórmula φ está en **forma normal negativa** si y sólo si cumple las siguientes condiciones:

- ★ φ **no contiene símbolos de condicional** ($\rightarrow$) ni **bicondicional** ($\leftrightarrow$)
- ★ Toda **negación** que aparece en φ afecta únicamente a **fórmulas atómicas**.

La Forma Normal Negativa (FNN) ya fue estudiada previamente en el contexto de la lógica proposicional, donde aprendimos a eliminar condicionales y bicondicionales, y a empujar las negaciones hasta que afecten únicamente a fórmulas atómicas. En lógica de primer orden, el principio es el mismo: buscamos una representación en la que sólo aparezcan los conectivos $\wedge$, $\vee$ y $\neg$, con la restricción de que toda negación debe aplicarse directamente sobre un predicado. Para ello, primero transformamos la fórmula usando equivalencias lógicas para eliminar $\rightarrow$ y $\leftrightarrow$, y luego aplicamos las leyes de De Morgan y la doble negación. Esta forma es esencial para facilitar transformaciones posteriores, como el paso a la forma prenex y eventualmente a la forma clausular, que requieren una estructura más uniforme y manejable.

Ejemplo 2

Sea φ la fórmula definida en el Ejemplo 1. Encontrar $\text{fnn}(\text{rec}(\varphi))$, es decir, la Forma Normal Negativa de la fórmula ya rectificada.

Solución:

$$\begin{aligned} \text{fnn}(\text{rec}(\varphi)) &\equiv \forall x \exists y \neg (P(x, y) \vee \neg Q(x) \rightarrow \exists w \neg T(a, w)) \\ &\equiv \forall x \exists y (P(x, y) \vee \neg Q(x) \wedge \neg(\exists w \neg T(a, w))) \\ &\equiv \forall x \exists y (P(x, y) \vee \neg Q(x) \wedge \forall w \neg \neg T(a, w)) \\ &\equiv \forall x \exists y (P(x, y) \vee \neg Q(x) \wedge \forall w T(a, w)) \end{aligned}$$

En la primera línea, eliminamos el condicional utilizando la equivalencia $A \rightarrow B \equiv \neg A \vee B$, junto con el manejo adecuado de la negación externa. Luego, en la segunda línea, se aplica la transformación de negación sobre un cuantificador existencial: $\neg \exists w \psi \equiv \forall w \neg \psi$. En la tercera línea, se elimina la doble negación utilizando la equivalencia $\neg \neg \psi \equiv \psi$. La fórmula resultante ya está en Forma Normal Negativa, pues no contiene condicionales ni bicondicionales, y todas las negaciones aplican únicamente sobre fórmulas atómicas.

Forma normal prenex

La **Forma Normal Prenex** consiste en reescribir una fórmula lógica de manera que **todos los cuantificadores queden al inicio**, como una secuencia ordenada, dejando el cuerpo de la fórmula libre de cuantificadores. Es decir, transforma una fórmula dada en una fórmula lógicamente equivalente de la forma:

$$Q_1x_1Q_2x_2\dots Q_nx_n\varphi$$

donde cada Q_i es un cuantificador ($\forall$ o $\exists$) y φ es una fórmula **sin cuantificadores** (también llamada matriz o parte cuantificada).

Definición 3. (Forma Normal Prenex)

Una fórmula φ está en **Forma Normal Prenex** si y sólo si tiene la forma:

$$Q_1x_1Q_2x_2\dots Q_nx_n\psi$$

donde cada $Q_i \in \{\forall, \exists\}$ para toda $1 \leq i \leq n$, y ψ es una fórmula **sin cuantificadores**. En este caso la secuencia $Q_1x_1Q_2x_2\dots Q_nx_n$ se denomina el **prefijo** de φ , y la subfórmula ψ se conoce como la **matriz** de φ .

La Forma Normal Prenex permite reorganizar las fórmulas de la lógica de predicados para que todos los cuantificadores aparezcan al inicio, lo cual facilita la aplicación de técnicas automáticas como la skolemización y la resolución. Esta transformación no cambia el significado lógico de la fórmula, pero la convierte en una estructura estándar compuesta por un prefijo (donde se encuentran todos los cuantificadores) y una matriz sin cuantificadores. Al separar explícitamente ambas partes, se simplifican muchos procesos de inferencia, verificación y traducción a otros formatos lógicos.

Algunas equivalencias útiles para la transformación a Forma Normal Prenex:

Si $x \notin$ las variables libres de φ y $\star \in \{\wedge, \vee, \rightarrow\}$, entonces se tienen las siguientes equivalencias:

$$\varphi \star \forall x \psi \equiv \forall x (\varphi \star \psi)$$

$$\forall x (\varphi \rightarrow \psi) \equiv \exists x (\varphi \rightarrow \psi)$$

$$\varphi \star \exists x \psi \equiv \exists x (\varphi \star \psi)$$

$$\exists x (\varphi \rightarrow \psi) \equiv \forall x (\varphi \rightarrow \psi)$$

Denotamos a la transformación de una fórmula φ a su Forma Normal Prenex como $\mathbf{fnp}(\varphi)$.

Ejemplo 3

Sea φ la fórmula obtenida en el Ejemplo 2. Encontrar $\mathbf{fnp}(\varphi)$, es decir, la transformación de φ a Forma Normal Prenex.

Solución:

$$\begin{aligned} \mathbf{fnp}(\varphi) &\equiv \forall x \exists y ((P(x, y) \vee \neg Q(x)) \wedge (\forall w T(a, w))) \\ &\equiv \forall x \exists y \forall w ((P(x, y) \vee \neg Q(x)) \wedge T(a, w)) \end{aligned}$$

En la primera línea, identificamos que el cuantificador $\forall w$ aparece dentro de una conjunción. Como

la variable w no aparece libre en la subfórmula $P(x, y) \vee \neg Q(x)$, es posible aplicar una equivalencia válida para mover el cuantificador al frente. En la segunda línea, se reescribe la fórmula agrupando todos los cuantificadores al inicio, obteniendo así su Forma Normal Prenex. Esta forma facilita procedimientos como la skolemización, ya que separa claramente el prefijo de cuantificadores de la matriz libre de cuantificadores.

Forma Normal de Skolem

La **Forma Normal de Skolem** tiene como objetivo eliminar los cuantificadores existenciales mediante una técnica conocida como **skolemización**. Esta técnica consiste en reemplazar las variables existenciales por **testigos** adecuados, es decir, por términos que garantizan la existencia planteada por el cuantificador. El procedimiento se realiza de la siguiente manera:

$$\begin{array}{l} \exists x \varphi \text{ se reemplaza por } \varphi[x := c] \\ \forall x_1 \dots \forall x_n \exists y \varphi \text{ se reemplaza por } \forall x_1 \dots \forall x_n \varphi[y := g(x_1, \dots, x_n)] \end{array}$$

Donde c es un nuevo **símbolo de constante**, llamado **constante de Skolem**, y g es un nuevo **símbolo de función**, conocido como **función de Skolem**.

Definición 4. (Forma Normal de Skolem)

Una fórmula está en **Forma Normal de Skolem** si tiene la forma:

$$\forall x_1 \dots \forall x_n \varphi$$

donde φ es una fórmula **sin cuantificadores** y está escrita en **forma normal conjuntiva (FNC)**, es decir, como una conjunción de cláusulas.

La Forma Normal de Skolem se obtiene al aplicar una serie de pasos sistemáticos: rectificación, forma normal negativa, forma prenex y skolemización. El resultado es una fórmula donde todos los cuantificadores están al inicio (y son exclusivamente universales) y la parte restante (la matriz) ya no contiene cuantificadores y está organizada como una conjunción de disyunciones de literales. Esta forma es especialmente útil porque estandariza la representación de fórmulas en lógica de primer orden. Cabe recordar que ya hemos estudiado la forma normal conjuntiva (FNC) en lógica proposicional, y en este contexto simplemente la aplicamos a la matriz una vez que se han eliminado los cuantificadores existenciales mediante funciones o constantes de Skolem.

Representamos a la Forma Normal de Skolem de una fórmula φ como $\mathbf{fns}(\varphi)$.

Ejemplo 4

Sea φ la fórmula obtenida en el Ejemplo 3. Encontrar $\mathbf{fns}(\varphi)$, es decir, su Forma Normal de Skolem.

Solución:

$$\begin{aligned} \mathbf{fns}(\varphi) &\equiv \forall x (\exists y (\forall w ((P(x, y) \vee \neg Q(x)) \wedge T(a, w)))) \\ &\equiv \forall x \forall w ((P(x, f(x)) \vee \neg Q(x)) \wedge T(a, w)) \end{aligned}$$

Para obtener la Forma Normal de Skolem, identificamos el cuantificador existencial $\exists y$ dentro del alcance del cuantificador universal $\forall x$. Como y depende de x , se sustituye por una función de Skolem $f(x)$, que actúa como testigo. Luego, se eliminan los cuantificadores existenciales y se reordenan los cuantificadores universales al inicio. El resultado es una fórmula equivalente en la que todos los cuantificadores son universales y la parte restante (la matriz) ya no contiene cuantificadores. Esta estructura es necesaria antes de convertir la fórmula a forma clausular.

Forma clausular

La **Forma Clausular** se obtiene, al igual que en la lógica proposicional, a partir de la **matriz** de la fórmula en Forma Normal de Skolem.

Definición 5. (Forma Clausular)

Sean φ y una fórmula $\mathbf{fns}(\varphi) = \forall x_1 \dots \forall x_n \psi$ su Forma Normal de Skolem. Si la matriz ψ es una conjunción de cláusulas, es decir,

$$\psi = C_1 \wedge \dots \wedge C_k$$

entonces la **forma clausular** de φ , denotada Cl es la secuencia de cláusulas:

$$Cl(\varphi) = C_1 \wedge \dots \wedge C_k$$

donde, sin pérdida de generalidad, se asume que las variables de cláusula son distintas de la que aparecen en las demás. Es decir, **todas las cláusulas tienen variables ajenas entre sí**.

Esta definición formaliza el paso final del proceso de transformación de una fórmula en lógica de predicados hacia una representación lista para métodos automáticos como la resolución. Partimos de la Forma Normal de Skolem, que ya no contiene cuantificadores existenciales, y extraemos de su matriz las cláusulas que forman la base del razonamiento mecanizado. Hacemos que las variables de cada cláusula sean distintas de las de las demás para evitar conflictos durante procesos como la unificación, y porque en lógica de primer orden las variables se consideran localmente ligadas dentro de cada cláusula. Esta normalización permite operar sobre las cláusulas como unidades independientes.

Ejemplo 5

Sea φ la fórmula obtenida en el Ejemplo 4. Encontrar $Cl(\varphi)$, es decir, la forma clausular de φ .

Solución:

$$\mathbf{fns}(\varphi) \equiv \forall x \forall w ((P(x, f(x)) \vee \neg Q(x)) \wedge T(a, w))$$

La matriz de esta fórmula es una conjunción de dos cláusulas:

$$(P(x, f(x)) \vee \neg Q(x)) \wedge T(a, w)$$

Por lo tanto, la forma clausular de φ es la siguiente secuencia de cláusulas:

$$\text{Cl}(\varphi) = P(x, f(x)) \vee \neg Q(x), \quad T(a, w)$$

En la Forma Normal de Skolem, los cuantificadores universales se eliminan al pasar a la forma clausular, ya que las variables que cuantifican se entienden como variables libres con alcance local en cada cláusula. Extraemos las cláusulas directamente de la matriz, que ya está en forma normal conjuntiva. Cada conjunción da lugar a una cláusula en la secuencia, y el resultado es una colección de fórmulas disyuntivas que pueden tratarse como unidades independientes en procesos de deducción automática.

3. Unificación

Hasta este punto hemos recorrido una serie de transformaciones sobre fórmulas de la lógica de predicados: desde la rectificación, pasando por la forma normal negativa, la forma prenex, la skolemización y finalmente la forma clausular. Estas transformaciones tienen como propósito preparar las fórmulas para ser utilizadas en procedimientos de inferencia automática, en particular, en la aplicación de la **regla de resolución binaria** que estudiaremos en breve.

Para poder aplicar esta técnica en el contexto de la lógica de predicados, es necesario contar con una herramienta que nos permita comparar y hacer compatibles los términos que aparecen en diferentes cláusulas. Esa herramienta es la **unificación**. El proceso de unificación consiste en encontrar, dado un conjunto de literales W , una sustitución σ tal que al aplicar σ a cada elemento de W (es decir $W\sigma$), se obtenga un conjunto con un único literal. En esta sección se revisarán algunos conceptos fundamentales relacionados con la unificación, los cuales serán esenciales para entender y aplicar posteriormente la regla de resolución. El algoritmo de unificación que utilizaremos está **inspirado en el original de Martelli-Montanari**, pero se presenta en una versión más accesible y orientada a la enseñanza, siguiendo la propuesta de **Shriram Krishnamurthi**.

Unificación de literales

Se dice que una sustitución σ es un **unificador** de dos literales ℓ_1 y ℓ_2 si, al aplicarla a ambos, se obtiene la misma fórmula de primer orden; es decir, $\ell_1\sigma = \ell_2\sigma$.

Ejemplo 6

Sea la sustitución $\sigma = [x, y := a, a]$ y los términos $\ell_1 = P(x, f(y))$ y $\ell_2 = P(x, f(x))$.

Aplicación de la sustitución:

$$\ell_1\sigma = P(a, f(a)) \quad \text{y} \quad \ell_2\sigma = P(a, f(a))$$

Conclusión:

Como la aplicación de σ a ambos literales produce el mismo término, se concluye que σ es un **unificador** de ℓ_1 y ℓ_2 . Este ejemplo ilustra cómo una misma sustitución puede resolver las diferencias

estructurales entre dos términos mediante la asignación adecuada de constantes a variables.

Composición de sustituciones

Es posible encontrar unificadores de manera algorítmica, y una de las operaciones fundamentales para ello es la **composición de sustituciones**, cuya propiedad principal se establece en la siguiente proposición.

Proposición (Composición de sustituciones)

Sean

$$\sigma = [x_1 := t_1, \dots, x_n := t_n] \quad \text{y} \quad \tau = [y_1 := s_1, \dots, y_m := s_m]$$

dos sustituciones. La **composición** $\sigma \circ \tau$ (también denotada $\sigma\tau$) es la sustitución definida por:

$$\sigma\tau = \{x_i := t_i\tau \mid t_i\tau \neq x_i\} \cup \{y_j := s_j \mid y_j \notin \{x_1, \dots, x_n\}\}$$

Demostración.

Queremos construir una sustitución $\sigma\tau$ tal que, para todo término t , se cumpla:

$$t(\sigma\tau) = (t\sigma)\tau$$

Es decir, aplicar la composición a un término equivale a aplicar primero σ y luego τ .

Para ello, definimos $\sigma\tau$ como el conjunto de pares variable-término que satisfacen las siguientes condiciones:

- ★ Para cada par $x_i := t_i \in \sigma$, se incluye $x_i := t_i\tau$ en $\sigma\tau$, a menos que $t_i\tau = x_i$, en cuyo caso se omite por redundancia. Esto asegura que $x_i(\sigma\tau) = (x_i\sigma)\tau$.
- ★ Para cada par $y_j := s_j \in \tau$ tal que $y_j \notin \{x_1, \dots, x_n\}$, se incluye directamente $y_j := s_j$ en $\sigma\tau$, ya que σ no afecta a y_j .

Con esta construcción, y por inducción estructural sobre los términos, se puede verificar que para todo término t , se cumple:

$$t(\sigma\tau) = (t\sigma)\tau$$

lo cual caracteriza la composición de sustituciones como la aplicación secuencial de dos sustituciones. □

La composición de sustituciones nos permite combinar dos sustituciones de forma coherente, respetando el orden de aplicación. Esta operación es esencial en algoritmos de unificación, ya que a medida que se resuelven nuevas igualdades entre términos, las sustituciones se acumulan y deben aplicarse de forma consistente sobre todas las fórmulas pendientes. La definición asegura que no se repitan asignaciones innecesarias y que se mantenga la independencia de variables, lo que facilita la implementación y el análisis del algoritmo.

Ejemplo 7

Dadas las siguientes sustituciones:

$$\sigma = [x, y := f(y), w] \quad \rho = [x, z := g(w), b] \quad \tau = [y, w, v := b, f(c), w]$$

Composición paso a paso:

1. $\sigma \circ \rho$

★ Aplicamos ρ a cada término del lado derecho de σ :

$$\sigma = [x := f(y), y := w] \Rightarrow \sigma\rho = [x := f(y), y := b]$$

(Obsérvese que w se reemplaza por b en $y := w$, porque $w \in \text{dom}(\rho)$).

★ Luego se agregan las sustituciones de ρ para variables que no están en σ :

$$z := b$$

★ Resultado:

$$\sigma\rho = [x := f(y), y := b, z := b]$$

2. $\sigma \circ \tau$

★ Aplicamos τ a cada término del lado derecho de σ :

$$\sigma = [x := f(y), y := w] \Rightarrow \sigma\tau = [x := f(b), y := f(c)]$$

★ Luego se agregan las sustituciones de τ para variables que no están en σ :

$$v := w$$

★ Resultado:

$$\sigma\tau = [x := f(b), y := f(c), v := w]$$

3. $\rho \circ \tau$

★ Aplicamos τ a cada término del lado derecho de ρ :

$$\rho = [x := g(w), z := b] \Rightarrow \rho\tau = [x := g(f(c)), z := b]$$

★ Luego se agregan las sustituciones de τ para variables que no están en ρ :

$$y := b, v := w$$

★ Resultado:

$$\rho\tau = [x := g(f(c)), z := b, y := b, v := w]$$

Comentario. Cada composición se realiza aplicando la segunda sustitución a los términos de la primera, y luego completando con las asignaciones que no se sobrescriben. El resultado es una nueva sustitución que representa la aplicación secuencial de ambas.

Observación 1

La proposición anterior nos proporciona una caracterización formal de la **composición de sustituciones**, operación fundamental para construir unificadores de manera acumulativa. Esta noción será

clave para formular una definición importante en lo que sigue. Sin embargo, gracias a la manera en que está diseñado el algoritmo de unificación que adoptamos (inspirado en la presentación de Shriram Krishnamurthi) no será necesario preocuparnos por aplicar explícitamente esta composición: el algoritmo la realiza de forma automática al modificar las sustituciones y la pila de restricciones en cada paso.

Unificador más general

Definición 6. (Unificador más general)

Sea W un conjunto de términos. Una sustitución σ es un **unificador más general** (umg) de W si cumple que, para toda sustitución τ que también unifica W , existe una sustitución ρ tal que:

$$\sigma\rho = \tau$$

El unificador más general es, por así decirlo, el *más flexible* o *menos comprometido* de todos los posibles unificadores. En lugar de imponer sustituciones específicas innecesarias, deja abiertas tantas variables como sea posible, permitiendo que cualquier otro unificador pueda obtenerse a partir de él mediante una composición adicional. Esta propiedad es fundamental porque garantiza que, si encontramos el unificador más general, podemos derivar a partir de él cualquier otro unificador que necesitemos, sin tener que buscarlos todos desde cero.

Dicho de otro modo, si tenemos una lista de unificadores de un conjunto W , y elegimos uno de ellos, digamos σ , entonces σ es un unificador más general si al componerlo con una sustitución distinta para cada uno de los demás, se obtienen exactamente esas sustituciones. Obsérvese que no existe un único unificador más general, ya que pueden existir varios que lo sean, todos equivalentes entre sí hasta renombramiento de variables.

Esta noción no sólo es útil en la resolución binaria, sino que también aparece en otras áreas de las ciencias de la computación, como en los sistemas de tipos de algunos lenguajes de programación, donde se utiliza para determinar el tipo más general que puede asignarse a una expresión sin perder corrección. Por ejemplo, si una función puede tomar como argumento cualquier tipo y devolver ese mismo tipo, su tipo más general sería algo como $\alpha \rightarrow \alpha$, donde α representa una variable de tipo. Aunque no entraremos en ese tema por ahora, vale la pena notar esta conexión.

El algoritmo

Para este algoritmo tenemos una pila de restricciones y un conjunto de sustituciones.

1. Si X y Y son constantes idénticas, no hagas nada.
2. Si X y Y son identificadores idénticos, no hagas nada.
3. Si X es un identificador, reemplaza, todas las apariciones de X por Y tanto en la pila como en las sustituciones, y añadimos $[X:=Y]$ en las sustituciones.
4. Si Y es un identificador, reemplaza, todas las apariciones de Y por X tanto en la pila como en las sustituciones, y añadimos $[Y:=X]$ en las sustituciones.

5. Si X es un constructor de la forma $C(X_1, \dots, X_n)$ para algún constructor C , y Y es de la forma $C(Y_1, \dots, Y_n)$, es decir, tienen el mismo constructor y el mismo número de argumentos, entonces, agrega $X_i = Y_i$ con $1 \leq i \leq n$ a la pila.
6. En cualquier otro caso, X y Y no unifican y se reporta un error.

En el contexto de la lógica de predicados, podemos interpretar los elementos mencionados en el algoritmo de la siguiente manera:

- ★ **Constantes** Constantes del conjunto de términos.
- ★ **Identificadores** Variables del conjunto de términos.
- ★ **Constructores** Corresponden a funciones o predicados aplicados a términos. En el algoritmo, lo que importa es su forma sintáctica: un símbolo aplicado a varios subcomponentes.

Observación 2

Es importante destacar que el algoritmo está formulado en una terminología general (constantes, identificadores, constructores) porque no está limitado al contexto de la lógica de predicados. También puede aplicarse en otros dominios, como sistemas de tipos, análisis de programas o programación lógica, donde los términos tienen estructuras similares aunque interpretaciones distintas. Esta abstracción permite usar el mismo algoritmo en una amplia variedad de aplicaciones.

Ejemplo 8

Supongamos las siguientes cláusulas:

$$\mathcal{C} = P(x, y) \vee \neg Q(a, x) \quad \mathcal{D} = R(z) \vee Q(z, b)$$

Observaciones:

- ★ Las cláusulas $\mathcal{C}$ y $\mathcal{D}$ tienen variables ajenas, es decir, no comparten variables.
- ★ En $\mathcal{C}$ identificamos la literal $\ell = \neg Q(a, x)$ y en $\mathcal{D}$ la literal $\ell' = Q(z, b)$.

Busquemos su unificador más general aplicando el algoritmo paso a paso:

Acción	Pila	Sustitución
Inicio	$Q(a, x) = Q(z, b)$	
5	$a = z$ $x = b$	
4	$x = b$	$[z := a]$
3		$[z := a]$ $[x := b]$

Conclusión: El unificador más general obtenido es $\sigma = [z := a, x := b]$. Este resultado nos permitirá aplicar la regla de resolución binaria sobre las cláusulas $\mathcal{C}$ y $\mathcal{D}$, ya que las literales $\neg Q(a, x)$ y $Q(z, b)$ se pueden hacer coincidir mediante σ .

Ejemplo 9

Consideremos el conjunto de términos:

$$W = \{Q(x, b, f(z)), Q(f(a), y, w), Q(x, b, f(c))\}$$

Aplicamos el algoritmo de unificación paso a paso:

Acción	Pila	Sustitución
Inicio	$Q(x, b, f(z)) = Q(f(a), y, w)$ $Q(f(a), y, w) = Q(x, b, f(c))$	
5	$x = f(a)$ $b = y$ $f(z) = w$ $Q(f(a), y, w) = Q(x, b, f(c))$	
3	$b = y$ $f(z) = w$ $Q(f(a), y, w) = Q(f(a), b, f(c))$	$[x := f(a)]$
4	$f(z) = w$ $Q(f(a), b, w) = Q(f(a), b, f(c))$	$[x := f(a), y := b]$
4	$Q(f(a), b, f(z)) = Q(f(a), b, f(c))$	$[x := f(a), y := b, w := f(z)]$
5	$f(a) = f(a)$ $b = b$ $f(z) = f(c)$	$[x := f(a), y := b, w := f(z)]$
5	$a = a$ $b = b$ $f(z) = f(c)$	$[x := f(a), y := b, w := f(z)]$
1	$b = b$ $f(z) = f(c)$	$[x := f(a), y := b, w := f(z)]$
1	$f(z) = f(c)$	$[x := f(a), y := b, w := f(z)]$
5	$z = c$	$[x := f(a), y := b, w := f(z)]$
3		$[x := f(a), y := b, w := f(c), z := c]$

Conclusión: El conjunto es unificable y el unificador más general obtenido es:

$$\sigma = [x := f(a), y := b, w := f(c), z := c]$$

Ejemplo 10

Consideremos el siguiente conjunto de cláusulas:

$$W = \{P(f(a, z), r, c), P(f(y, b), b, x), P(w, z, t)\}$$

Aplicamos el algoritmo paso a paso para encontrar un unificador:

Acción	Pila de restricciones	Sustitución
Inicio	$P(f(a, z), r, c) = P(f(y, b), b, x)$ $P(f(y, b), b, x) = P(w, z, t)$	
5	$f(a, z) = f(y, b)$ $r = b$ $c = x$ $P(f(y, b), b, x) = P(w, z, t)$	
5	$a = y$ $z = b$ $r = b$ $c = x$	$[y := a]$
4	$P(f(a, b), b, x) = P(w, z, t)$	$[y := a, z := b]$
3	$r = b, c = x$	$[y := a, z := b, r := b]$
3	$c = x$	$[y := a, z := b, r := b, x := c]$
4	$P(f(a, b), b, c) = P(w, z, t)$	$[y := a, z := b, r := b, x := c]$
5	$f(a, b) = w$ $b = b$ $c = t$	$[y := a, z := b, r := b, x := c]$
4	$b = b, c = t$	$[y := a, z := b, r := b, x := c, w := f(a, b)]$
1	$c = t$	$[y := a, z := b, r := b, x := c, w := f(a, b)]$
3		$[y := a, z := b, r := b, x := c, w := f(a, b), t := c]$

Conclusión: El conjunto es unificable, y el unificador más general obtenido es:

$$\sigma = [y := a, z := b, r := b, x := c, w := f(a, b), t := c]$$

4. Resolución binaria con unificación

La **resolución binaria** es de gran importancia, ya que proporciona un método para decidir si $\Gamma \models \varphi$. Como ya vimos al estudiar lógica proposicional, esto se logra al derivar la **cláusula vacía** $\square$ a partir de las formas clausulares del conjunto $\Gamma \cup \{\neg\varphi\}$. Esta derivación indica que la negación de φ es incompatible con Γ , por lo que φ es una consecuencia lógica del conjunto.

La **regla de resolución binaria** se define de la siguiente manera:

- ★ Se tienen dos cláusulas $\mathcal{C}$ y $\mathcal{D}$ cuyas variables son ajenas entre sí.
- ★ Existe en $\mathcal{C}$ una literal ℓ y en $\mathcal{D}$ una literal ℓ' tales que ℓ^c (la literal complementaria de ℓ) es unificable con ℓ' ; es decir, existe un unificador más general σ .
- ★ La regla produce como resultado la cláusula $(\mathcal{C}_1 \vee \mathcal{D}_1)\sigma$, donde $\mathcal{C}_1$ y $\mathcal{D}_1$ son el resultado de eliminar ℓ y ℓ' respectivamente de $\mathcal{C}$ y $\mathcal{D}$, y luego aplicar la sustitución σ .

$$\frac{\mathcal{C} =_{def} \mathcal{C}_1 \vee \ell \quad \mathcal{D} =_{def} \mathcal{D}_1 \vee \ell' \quad Var(\mathcal{C}) \cap Var(\mathcal{D}) = \emptyset \quad \sigma \text{ umg de } \{\ell^c, \ell'\}}{(\mathcal{C}_1 \vee \mathcal{D}_1)\sigma}$$

Ejemplo 11

Verifiquemos que se cumple la siguiente consecuencia lógica:

$$\forall x \exists y (C(x) \rightarrow P(y) \wedge A(x, y)) \models \forall z (C(z) \rightarrow \exists w (P(w) \wedge A(z, w)))$$

Transformamos la premisa y la negación de la conclusión a forma clausular. El conjunto resultante es:

$$\left\{ \begin{array}{l} \neg C(x) \vee P(f(x)), \\ \neg C(x) \vee A(x, f(x)), \\ C(a), \\ \neg P(w) \vee \neg A(a, w) \end{array} \right\}$$

A continuación se muestra la derivación paso a paso mediante resolución binaria:

1. $\neg C(x) \vee P(f(x))$ Hipótesis
2. $\neg C(x) \vee A(x, f(x))$ Hipótesis
3. $C(a)$ Hipótesis
4. $\neg P(w) \vee \neg A(a, w)$ Hipótesis
5. $P(f(a))$ Resolución (1, 3) con $[x := a]$
6. $\neg A(a, f(a))$ Resolución (5, 4) con $[w := f(a)]$
7. $\neg C(a)$ Resolución (6, 2) con $[x := a]$
8. $\square$ Resolución (3, 7)

Conclusión: Como hemos obtenido la cláusula vacía $\square$, concluimos que la fórmula original es una consecuencia lógica válida: la premisa implica necesariamente la conclusión.

5. Automatización

El proceso de resolución binaria con unificación puede ser formalizado y automatizado mediante lenguajes de programación funcional como HASKELL. Gracias a su capacidad para manipular estructuras algebraicas, listas de reglas y sustituciones de manera declarativa, HASKELL resulta especialmente adecuado para modelar procedimientos lógicos como la aplicación de la regla de resolución y la búsqueda de unificadores.

A continuación se presentan algunas firmas de funciones que modelan las operaciones básicas necesarias para implementar el proceso de resolución binaria:

```
-- Representación de términos, literales y cláusulas
type Termino = ...
type Literal = ...
type Clausula = [Literal]
type BaseClausulas = [Clausula]

-- Unificación
unificar :: Literal -> Literal -> Maybe Sustitucion

-- Aplicación de la regla de resolución
resolver :: Clausula -> Clausula -> Maybe Clausula

-- Búsqueda de la cláusula vacía
resolucion :: BaseClausulas -> Bool
```

Estas funciones permiten modelar computacionalmente el procedimiento lógico que hemos desarrollado manualmente en esta nota. Por ejemplo, la función `unificar` implementa el algoritmo de unificación y devuelve una sustitución cuando existe. La función `resolver` toma dos cláusulas, encuentra una pareja de literales complementarias unificables, y devuelve su cláusula resolvente si es posible. Finalmente, la función `resolucion` ejecuta una estrategia de búsqueda sobre un conjunto de cláusulas para determinar si puede derivarse la cláusula vacía $\square$, indicando que una conclusión es consecuencia lógica de las premisas.

Esto nos permite verificar automáticamente argumentos como el siguiente:

$$\Gamma = \left\{ \begin{array}{l} \neg C(x) \vee P(f(x)) \\ \neg C(x) \vee A(x, f(x)) \\ C(a) \\ \neg P(w) \vee \neg A(a, w) \end{array} \right\} \Rightarrow \square$$

codificando `Gamma` como una lista de cláusulas en Haskell y evaluando:

```
resolucion gamma
```

Este enfoque automatizado no sólo verifica la validez de consecuencias lógicas, sino que también constituye una base para sistemas de deducción automatizada, asistentes de prueba o herramientas educativas que simulan paso a paso el razonamiento lógico. El hecho de que todo el procedimiento pueda ser descrito con funciones puras y estructuras algebraicas subraya la elegancia y el poder expresivo de la programación funcional en contextos formales.

6. Conclusión

A lo largo de esta nota hemos recorrido el proceso completo que permite aplicar la regla de resolución binaria en lógica de predicados, desde la transformación sistemática de fórmulas a su forma clausular, hasta el estudio detallado del mecanismo de unificación. Hemos visto cómo cada etapa (rectificación, forma normal negativa, forma prenex, skolemización) no sólo tiene una justificación teórica, sino que es necesaria para garantizar que las fórmulas estén en una estructura que permita operar sobre ellas de forma sintáctica y automatizada. La unificación, en particular, nos ha mostrado cómo se pueden identificar patrones comunes entre términos complejos, permitiendo resolver inferencias que van más allá de la coincidencia literal. Finalmente, exploramos cómo estos procedimientos pueden ser implementados computacionalmente en lenguajes funcionales como HASKELL, lo que abre la puerta a desarrollar herramientas prácticas de deducción automatizada. Comprender estos conceptos no sólo es esencial para el estudio de la lógica computacional, sino que también sienta las bases para disciplinas como la programación lógica, los lenguajes formales y la verificación automática de programas.

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] Krishnamurthi, S. (2007). *Programming Languages: Application and Interpretation*.
- [4] Enderton, H. B. (2001). *A Mathematical Introduction to Logic* (2nd ed.). Elsevier.
- [5] Lloyd, J. W. (1987). *Foundations of Logic Programming* (2nd ed.). Springer-Verlag.
- [6] Baader, F., & Nipkow, T. (1998). *Term Rewriting and All That*. Cambridge University Press.
- [7] Thompson, S. (2011). *Haskell: The Craft of Functional Programming* (3rd ed.). Addison-Wesley.