2 PROGRAMACIÓN LÓGICA

Este capítulo introduce los fundamentos de la Programación Lógica. En particular, nos interesan los elementos de la Lógica de Primer Orden, necesarios para abordar la **resolución** como regla de inferencia única de una forma restringida de Lógica de Primer Orden, que nos permite automatizar su uso en el lenguaje de programación Prolog [8, 17]. Para ello, comenzaremos por introducir la Lógica de Primer Orden como un sistema formal, con su lenguaje y semántica característicos. Por lo tanto, definiremos la sintaxis del lenguaje de la Lógica de Primer Orden; así como la teoría del modelo y prueba detrás de la semántica de esta lógica. Posteriormente, restringiremos la Lógica de Primer Orden al caso de los Programas Definitivos, para introducir la regla de inferencia conocida como Resolución-SLD. Revisaremos en detalle los conceptos de substitución y unificación de variables; e implementaremos un algoritmo de unificación como el usado en la regla de resolución.

Resolución

2.1 LÓGICA DE PRIMER ORDEN

La lógica proposicional nos permite expresar conocimiento sobre situaciones que son de nuestro interés, mediante **enunciados declarativos**. Decimos que estos enunciados son declarativos en el sentido lingüístico del término, esto es, se trata de expresiones del lenguaje natural que son o bien verdaderas, o bien falsas; en contraposición a los enunciados imperativos e interrogativos. La Lógica proposicional es declarativa en este sentido, las proposiciones representan **hechos** que se dan o no en la realidad.

Enunciados declarativos

Hechos

La Lógica de Primer Orden que introduciremos en este capítulo, tienen un compromiso ontólogico más fuerte [77], donde la realidad implica, **objetos** y **relaciones** entre ellos. Consideren los siguientes ejemplos de enunciados declarativos:

Objetos y relaciones

1. Julia es madre y Luis es hijo de Julia.
2. Toda madre ama a sus hijos.

donde el enunciado (1) se refiere a los objetos de discurso *Julia* y *Luis*, usando propiedades de estos objetos, como ser *madre*; así como relaciones entre éstos, como ser *hijo* de alguien. El enunciado (2) se refiere a relaciones que aplican a todas las madres, en tanto que objetos de discurso. A esto nos referimos cuando hablamos de **representación** de un problema en el contexto de la Programación Lógica, a describir una situación en términos de objetos y relaciones entre ellos.

Representación

Al igual que en el caso proposicional, si se aplican ciertas reglas de razonamiento a tales representaciones, es posible obtener nuevas conclusiones. Por ejemplo, conociendo (1) y (2) es posible **inferir** (vía una versión de *Modus Ponens* un poco distinta de la proposicional) que:

Inferencia

3. Julia ama a Luis.

Universo de discurso

Funciones

De forma que, sería deseable poder describir los objetos que conforman un **universo de discurso**, personas en el ejemplo; así como las relaciones que se dan entre ellos, *hijo* y *madre* en el ejemplo; y computar tales descripciones para obtener conclusiones como (3). Al describir el problema que queremos resolver, también podemos hacer uso de **funciones**, relaciones en las cuales uno o más objetos del universo de discurso se relacionan con un objeto único. Por ejemplo, "madre de" puede representarse como una función (todo hijo tiene una sola madre genética), pero "hijo de" no. Esto se ilustra en la Figura 2.1.

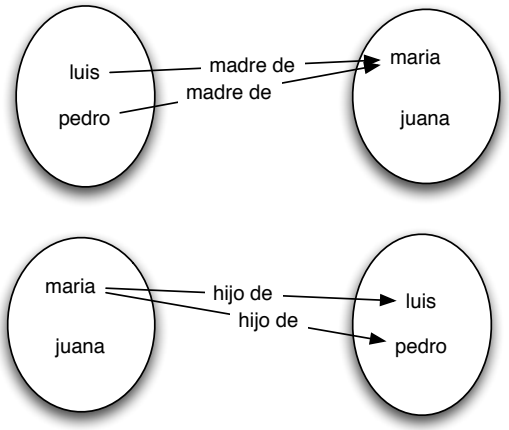


Figura 2.1: La relación *madre de* es una función; mientras que *hijo de* no lo es.

Sistemas formales

Como en todo **sistema formal**, será necesario especificar cuidadosamente los siguientes elementos:

- **Lenguaje.** Este elemento está asociado a la sintaxis de la Lógica de primer orden. El lenguaje de un sistema formal está dado por un conjunto de símbolos conocido como **alfabeto** y una serie de **reglas de sintácticas**. Una expresión es cualquier secuencia de símbolos pertenecientes al alfabeto. Cualquier expresión es, o no es, una **fórmula bien formada** (fbf). Las fórmulas bien formadas son las expresiones que pueden formarse con los símbolos del alfabeto a partir de las reglas de construcción ó sintácticas, que inducen una estructura de árbol en las fbf del lenguaje.

- **Teoría de modelo.** Este elemento está asociado a la semántica de la Lógica de Primer Orden. La teoría del modelo establece la interpretación de las fbfs en un sistema formal. Su función es relacionar las fbfs con alguna representación simplificada de la realidad que nos interesa, para establecer cuando una fbf es falsa y cuando verdadera. Esta versión de realidad corresponde a lo que informalmente llamamos "modelo". Sin embargo, en lógica, el significado de "modelo" está íntimamente relacionado con el lenguaje del sistema formal: Si la interpretación M hace que la fbf α [1] sea verdadera, se dice que M es un **modelo** de α

1 Los caracteres griegos se usan como meta variables, es decir, no pertenecen al lenguaje de la Lógica de Primer Orden; se usan para hablar acerca de este lenguaje.

o que M **satisface** α, y se escribe $M \models \alpha$. Una fbf es **válida** si toda interpretación es un modelo para ella.

- **Teoría de prueba.** Este elemento está asociado con el razonamiento deductivo. La teoría de la prueba tiene como objetivo hacer de cada enunciado matemático, una fórmula demostrable y rigurosamente deducible. Para ello, la actividad matemática debería quedar reducida a la manipulación de símbolos y sucesiones de símbolos regulada por un conjunto de instrucciones dadas al respecto. La construcción de tal teoría implica, además del lenguaje del sistema formal, un subconjunto de fbf que tendrán el papel **axiomas** en el sistema, y un conjunto de **reglas de inferencia** que regulen diversas operaciones sobre los axiomas. Las fbf obtenidas mediante la aplicación sucesiva de las reglas de inferencia a partir de los axiomas se conocen como **teoremas** del sistema. Si una fbf α es derivable en el sistema formal, escribimos $\vdash \alpha$. La expresión $\Delta \vdash \alpha$ expresa que la fbf α es derivable a partir del conjunto de fbfs Δ.

A continuación abordaremos cada uno de estos elementos en detalle.

2.1.1 Lenguaje

Objetos

Universo de discurso

Básicamente, el lenguaje de la Lógica de Primer Orden introduce un conjunto de símbolos que nos permiten expresarnos acerca de los **objetos** en un dominio de discurso dado. El conjunto de todos estos objetos se conoce como **Dominio** o **Universo de discurso** ($\mathcal{U}$). Los miembros del universo de discurso pueden ser objetos concretos, p. ej., un libro, un robot, etc; abstractos, p. ej., números; e incluso, ficticios, p. ej., unicornios, etc. Así, un objeto es algo sobre lo cual queremos expresarnos. En esta sección utilizaremos como dominio el multi citado mundo de los bloques [30] que se muestra en la figura 2.2. El universo de discurso para tal escenario es el conjunto que incluye los cinco bloques, la el brazo robótico y la mesa: $\mathcal{U} = \{a, b, c, d, e, brazo, mesa\}$.

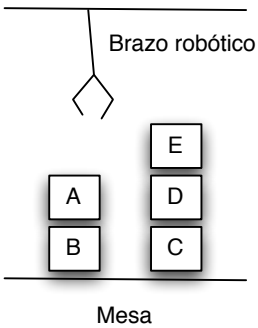


Figura 2.2: *El mundo de los bloques, usado en los ejemplos subsiguientes.*

Funciones

Una **función** es un tipo especial de relación entre los objetos del dominio de discurso. Este tipo de relaciones mapea un conjunto de objetos de entrada a un objeto único de salida. Por ejemplo, es posible definir la función parcial *sombrero* que mapea un bloque al bloque que se encuentra encima de él, si tal bloque existe. Las parejas correspondientes a esta función parcial, dado

el escenario mostrado en la figura 2.2 son: $\{(b,a),(c,d),(d,e)\}$. El conjunto de todas las funciones consideradas en la conceptualización del mundo se conoce como **base funcional**.

Base funcional

Un segundo tipo de relación sobre los objetos del dominio de discurso son los **predicados**. Diferentes predicados pueden definirse en el mundo de los bloques, por ejemplo, el predicado *sobre* que se cumple para dos bloques, si y sólo si el primero está inmediatamente encima del segundo. Para la escena mostrada en la figura 2.2, $sobre/2$ se define por los pares $\{(a,b),(d,c),(e,d)\}$. Otro predicado puede ser $libre/1$, que se cumple para un bloque si y sólo si éste no tiene ningún bloque encima. Este predicado tiene los siguientes elementos $\{a,e\}$. El conjunto de todos los predicados usados en la conceptuación se conoce como **base relacional**.

Predicados

Base relacional

Para universos de discurso finitos, existe un límite superior en el número posible de predicados n-arios que pueden ser definidos. Para un universo de **cardinalidad** b (cardinalidad es el número de elementos de un conjunto), existen b^n distintas n-tuplas. Cualquier predicado n-ario está definido por un subconjunto de estas b^n tuplas. Por lo tanto, la definición de un predicado n-ario debe corresponder a uno de los $2^{(b^n)}$ conjuntos posibles.

Cardinalidad

Además de las funciones y predicados, la flexibilidad de la Lógica de Primer Orden resulta del uso de variables y cuantificadores. Las **variables**, cuyos valores son objetos del universo de discurso, se suelen representar por cualquier secuencia de caracteres que inicie con una mayúscula [2]. El **cuantificador "para todo"** ($\forall$) nos permite expresar hechos acerca de todos los objetos en el universo del discurso, sin necesidad de enumerarlos. Por ejemplo, toda madre ... El **cuantificador "existe"** ($\exists$) nos permite expresar la existencia de un objeto en el universo de discurso con cierta propiedad en particular, por ejemplo, $\exists X\ libre(X) \wedge enLaMesa(X)$ expresa que hay al menos un objeto que no tiene bloques sobre él y aue se encuentra sobre la mesa. Revisemos estos conceptos formalmente.

Variables

Cuantificadores

El **alfabeto** de la Lógica de Primer Orden se obtiene al extender la Lógica Proposicional con un conjunto numerable de símbolos de predicados ($Pred$) y funciones ($Func$). Se asume un conjunto infinito de variables (Var) que toman valores en el universo de discurso. $|f|$ denota la **aridad** del predicado o función f, es decir, su número de argumentos.

Alfabeto

Aridad

Los componentes de nuestro lenguaje que nos permiten denotar objetos del universo de discurso se conocen como **términos**; y en la Lógica de Primer Orden se forman con variables, constantes y funciones aplicadas a arguments, que a su vez son términos. Por ejemplo, $calif(hermano(alex), sma)$ es un término que denota la calificación obtenida por el hermano de Álex en el curso de Sistemas Multi-Agentes, es decir un entero entre cero y cien. Utilizando notación BNF:

Términos

Definición 2.1 (Términos)**.** *Un término se define como:*

$$t ::= x \mid c \mid f(t,\ldots,t)$$

donde $x \in Var$; $c \in Func$ *tal que* $|c| = 0$; *y* $f \in Func$ *tal que* $|f| > 0$.

2 En algunos textos encontrarán que las variables se representan como cadenas de texto de inician con minúscula y los símbolos de predicado y funciones, con mayúsculas. He optado por lo opuesto, para seguir desde ahora la notación usada en Prolog.

Observen que los constructores básicos de los términos son las variables y las funciones. Las funciones de aridad cero se asumen como **constantes**. Se pueden formar términos más complejos usando functores de aridad mayor a cero, cuyos argumentos son a su vez términos.

Constantes

Definición 2.2 (Sintaxis)**.** *Las fbf del lenguaje de la Lógica de Primer Orden se construye a partir de las variables Var, los functores Func y los predicados Pred como sigue:*

$$\phi ::= P(t_1, \ldots, t_n) \mid \neg(\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid (\forall x\ \phi) \mid (\exists x\ \phi)$$

donde $P \in Pred$ es un símbolo de predicado de aridad $n > 0$; t_i denota términos sobre Func; y $x \in Var$.

Es posible adoptar como fbf a los predicados de aridad cero, que entonces se asumen como **variables proposicionales**. Esto tiene sentido si observamos que las fbf de la lógica de Primer Orden denotan enunciados declarativos sobre la relaciones que se dan entre los objetos del universo de discurso. De forma que las variables proposicionales son enunciados declarativos donde no nos interesa referirnos explicitamente a los objetos y sus relaciones, pero si posiblemente a su valor de verdad. La Lógica de Primer Orden subsume a la proposicional.

Variables proposicionales

Asumiremos que los cuantificadores tienen la misma **precedencia** que la negación. El resto de los operadores tiene la misma precedencia que en la lógica proposicional. Las fbf de la lógica de Primer Orden pueden representarse como árboles sintácticos. Por ejemplo, el árbol de la fbf $\forall X((p(X) \rightarrow q(X)) \wedge s(X, Y))$ se muestra en la figura 2.3.

Precedencia de los operadores

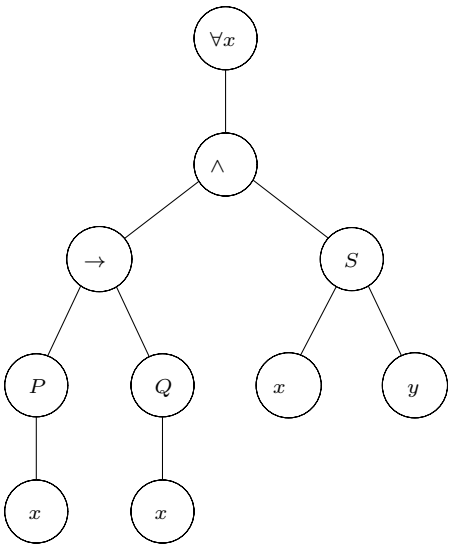


Figura 2.3: El árbol sintáctico de una fbf de la lógica de Primer Orden.

2.1.2 Semántica

En la Lógica Proposicional, una interpretación es una función que asigna valores de verdad a las variables proposicionales de una fbf. En la lógica

de Primer Orden, el concepto análogo debe asignar valores de verdad a las fórmulas atómicas del lenguaje; pero esto involucra normalmente la substitución de variables por objetos en el universo de discurso, de forma que las fbf puedan ser interpretadas como relaciones sobre $\mathcal{U}$ que se satisfacen.

Veamos un ejemplo intuitivo de este proceso, basado en el escenario del mundo de los bloques que se muestra en la Figura 2.2. Si queremos expresar que al menos algún bloque no tiene nada encima, podríamos usar los predicados *bloque* y *libre*, con su significado pretendido evidente, en la siguiente expresión: $\exists X\ bloque(X) \wedge libre(X)$. Esta fbf expresa que existe un X tal que X es un bloque y X está libre (no tiene otro bloque encima).

Cuando usamos cuantificadores, siempre tenemos en mente el universo de discurso, en este caso $\mathcal{U} = \{a, b, c, d, e, mesa, mano\}$. En el caso del predicado $bloque/1$ es de esperar que su **interpretación** sea un subconjunto de $\mathcal{U}$ como es el caso, ya que los objetos del universo de discurso que satisfacen la relación bloque son $\{a, b, c, d, e\} \subset \mathcal{U}$. En general, para un predicado de aridad n, su interpretación será un subconjunto de $\mathcal{U}^n$. Por ejemplo $sobre/2$, en la escena considerada, tiene una interpretación $\{(a, b), (e, d), (d, c)\} \subset \mathcal{U}^2$. Para una función de aridad n la interpretación será un mapeo, posiblemente parcial, $\mathcal{U}^n \mapsto \mathcal{U}$, por ejemplo $sombrero/1$ tiene como interpretación $\{(b) \mapsto a, (d) \mapsto e, (c) \mapsto d\}$, de forma que podemos computar expresiones como $sombrero(b) \mapsto a$.

Interpretación

Para definir una interpretación en la lógica de Primer Orden necesitamos una tupla $\langle D, V \rangle$, donde D es el universo de discurso; y V es una función, tal que para cualquier predicado de aridad n se obtiene el conjunto de las n-tuplas que corresponden a la interpretación del predicado. Para una constante, la función V regresa la misma constante, ej. $V(a) = a$. Algunas veces la expresión $V(\alpha)$, se abrevia α^V. Una posible interpretación para la escena mostrada en al figura 2.2 y sus bases relacional y funcional es:

$$
\begin{aligned}
a^V &= a \\
b^V &= b \\
c^V &= c \\
d^V &= d \\
e^V &= e \\
sobre^V &= \{(a, b), (e, d), (d, c)\} \\
enLaMesa^V &= \{b, c\} \\
libre^V &= \{a, e\} \\
porEncima^V &= \{(a, b), (e, d), (e, c), (d, c)\} \\
sombrero^V &= \{(b) \mapsto a, (d) \mapsto e, (c) \mapsto d\}
\end{aligned}
$$

Todo esto puede especificarse formalmente con la siguiente regla semántica:

Definición 2.3 (Interpretación)**.** *Una interpretación V, con respecto a un dominio de discurso D es una función que satisface las siguientes propiedades: i) Si $\alpha \in Const$, entonces $V(\alpha) = \alpha$; ii) Si $\alpha \in Pred$ es de aridad $n > 0$, entonces $V(\alpha) \subseteq D^n$; iii) Si $\alpha \in Func$ es de aridad $n > 0$, entonces $V(\alpha) \subseteq D^n \mapsto D$.*

En lo que sigue, se asume que las variables en las fbf están **acotadas**, es

Variables libres y acotadas

decir, bajo el alcance de un cuantificador. En otro caso se dice que la variable es **libre**. Consideren la siguiente fbf cuyo árbol se muestra en la figura 2.3. Primero, los cuantificadores, al igual que la negación, tienen un solo sub-árbol sintáctico. Segundo, los predicados de aridad n tienen n sub-arboles. En un árbol sintáctico de una fbf las variables ocurren en dos sitios: al lado de un cuantificador y como hojas del árbol. Si subimos por el árbol a partir de las hojas etiquetadas con la variable X, llegaremos al nodo $\forall X$, por tanto decimos que las ocurrencias de X está acotadas por el cuantificador universal, o que X está bajo el **alcance** de $\forall X$; en el caso de la variable Y llegamos al mismo nodo, pero Y no tienen nada que ver con $\forall X$, por lo que decimos que es una variable libre.

Alcance

Definición 2.4 (Variable libre y acotada)**.** *Sea ϕ una fbf en la lógica de Primer Orden. Una ocurrencia de la variable X en ϕ es libre si X es un nodo hoja en el árbol sintáctico de ϕ tal que no hay caminos hacía arriba del árbol que lleven a un nodo etiquetado como $\forall X$ o $\exists X$. De otra forma se dice que la ocurrencia de la variable es acotada. Para las fbf $\forall X\phi$ y $\exists X\phi$ se dice que ϕ (menos toda sub-fórmula de ϕ, $\forall X\psi$ o $\exists X\psi$) es el alcance del cuantificador $\forall X$ y $\exists X$, respectivamente.*

Si una fbf no tiene variables libres, se dice que es una fbf **cerrada**. Si no tiene variables libres, ni acotadas, se dice que es una **fórmula de base**.

Fórmulas cerradas
Fórmula de base

Por razones que serán evidentes más adelante, conviene interpretar las variables de una fbf de forma separada. Una **asignación de variables** es una función de las variables del lenguaje a objetos en el universo de discurso. Decimos que U es una asignación de variables basada en el modelo $M = \langle D, V\rangle$ si para todo $\alpha \in Var$, $U(\alpha) \in D$. Al igual que en el caso de las interpretaciones α^U es una abreviatura de $U(\alpha)$. Por ejemplo, en lo que sigue a la variable X se le asigna el bloque a y a la variable Y el bloque b:

Asignación de variables

$$X^U = a$$
$$Y^U = b$$

Una interpretación V y una asignación de variables U pueden combinarse en una **asignación de términos**, donde los términos que no son variables son procesados por la interpretación, y las variables por la asignación de variables.

Asignación de términos

Definición 2.5 (Asignación de términos)**.** *Dadas una interpretación V y una asignación de términos U, la asignación de términos T_{VU} es un mapeo de términos a objetos en el universo de discurso, como sigue:*

1. *Si $\alpha \in Const$ entonces $T_{VU}(\alpha) = \alpha^V$*
2. *Si $\alpha \in Var$ entonces $T_{VU}(\alpha) = \alpha^U$*
3. *Si α es un término de la forma $\phi(\tau_1, \ldots, \tau_n)$; y $\phi^V = g$, mientras que $T_{VU}(\tau_i) = x_i$, entonces $T_{VU}(\alpha) = g(x_1, \ldots, x_n)$.*

Los conceptos de interpretación y asignación de variables son importantes porque nos permiten definir una noción relativa de verdad llamada **satisfacción**. El hecho de que la fbf ϕ se satisfaga en una interpretación V y una

Satisfacción

asignación de variables U se denota como $\models_I \phi[U]$. Si ese es el caso, se dice que ϕ es **verdadera** en relación a la interpretación V y la asignación de variables U. Como suele ser el caso, la definición de la relación de satisfacción depende de la forma de ϕ, así que se define por casos:

Definición 2.6 (Satisfacción)**.** *Dado un modelo $M = \langle D, V \rangle$ y una asignación de términos T_{VU}, la satisfacción se define como sigue:*

1. $\models_V (\phi = \psi)[U]$ *si y sólo si* $T_{VU}(\phi) = T_{VU}(\psi)$.
2. $\models_V \phi(\tau_1, \ldots, \tau_n)[U]$ *si y sólo si* $(T_{VU}(\tau_1), \ldots, T_{VU}(\tau_n)) \in \phi^V$.
3. $\models_V (\neg\phi)[U]$ *si y sólo si* $\not\models_V \phi[U]$.
4. $\models_V (\phi \wedge \psi)[U]$ *si y sólo si* $\models_V \phi[U]$ *y* $\models_V \psi[U]$.
5. $\models_V (\phi \vee \psi)[U]$ *si y sólo si* $\models_V \phi[U]$ *o* $\models_V \psi[U]$.
6. $\models_V (\phi \rightarrow \psi)[U]$ *si y sólo si* $\not\models_V \phi[U]$ *o* $\models_V \psi[U]$.
7. $\models_V (\forall\nu\phi)[U]$ *si y sólo si para todo* $d \in D$ *es el caso que* $\models_V \phi[W]$, *donde* $\nu^W = d$ *y* $\mu^W = \mu^U$ *para* $\mu \neq \nu$.
8. $\models_V (\exists\nu\phi)[U]$ *si y sólo si para algún* $d \in D$ *es el caso que* $\models_V \phi[W]$, *donde* $\nu^W = d$ *y* $\mu^W = \mu^U$ *para* $\mu \neq \nu$.

La igualdad de dos términos (1) se satisface si ambos denotan al mismo objeto bajo la asignación de términos. Una fbf atómica se satisface (2) si la tupla de sus argumentos bajo la asignación de términos está en la interpretación de su predicado. La regla de satisfacción para fbf cuantificadas universalmente (7) requieren que la fórmula ϕ se satisfaga para todas las asignaciones posibles de la variable cuantificada ν. Las asignaciones de variables que son idénticas para todas las variables, excepto posiblemente para ν, se dicen **ν-alternativas**. La regla de satisfacción para fbf cuantificadas existencialmente (8) requieren que la fórmula ϕ se satisfaga en al menos una asignación posible de la variable cuantificada ν. Las reglas de satisfacción para los operadores lógicos (3-6) son muy similares a los de la lógica proposicional.

Asignaciones alternativas

Como recordaran, de una interpretación V que satisface una fbf ϕ para toda asignación de variables, se dice que es un **modelo** de ϕ; lo cual se denota por $\models_V \phi$. Por ejemplo, siguiendo con la escena del mundo de los cubos: $\models_V enLaMesa(X) \vee \neg enLaMesa(X)$. Observen que la asignación de variables no tiene efectos en la satisfacción de una fbf cerrada o de base. Por consiguiente, toda interpretación que satisface una de estas fórmulas para una asignación de variables, es un modelo de la fórmula.

Modelo

Una fbf se dice **satisfacible** si y sólo si existe alguna interpretación y asignación de variables que la satisfacen. De otra forma se dice que la fórmula es insatisfacible. Una fbf es **válida** si y sólo si se satisface en toda interpretación y asignación de variables. Al igual que en el caso proposicional, las fórmulas válidas lo son en virtud de su estructura lógica y por tanto, no proveen información sobre el dominio descrito. Por ejemplo: $p(X) \vee \neg p(X)$ es una fbf válida.

Fórmula satisfacible

Fórmula válida

2.1.3 Inferencia

Volvamos al ejemplo de la introducción:

1. Toda madre ama a sus hijos.
2. Julia es madre y Luis es hijo de Julia.

Conociendo (1) y (2) es posible concluir que:

3. Julia ama a Luis.

Podemos formalizar este ejemplo en Lógica de Primer Orden como sigue:

1. $\forall X\ \forall Y\ madre(X) \wedge hijoDe(Y, X) \rightarrow ama(X, Y)$
2. $madre(julia) \wedge hijoDe(luis, julia)$
3. $ama(julia, luis)$

Una vez que hemos formalizado nuestros enunciados, el proceso de inferencia puede verse como un proceso de manipulación de fbf, donde a partir de formulas como (1) y (2), llamadas **premisas**, se produce la nueva fbf (3) llamada **conclusión**. Estas manipulaciones se pueden formalizar mediante **reglas de inferencia**. Al igual que en la lógica proposicional, es importante que en una consecuencia, la conclusión se siga de las premisas y las reglas de inferencia adoptadas. Entre las reglas de inferencia de la lógica de Primer Orden encontramos:

Reglas de inferencia

- **Modus Ponens.** O regla de eliminación de la implicación. Esta regla dice que siempre que las fbfs de la forma α y $\alpha \rightarrow \beta$ pertenezcan a las premisas o sean concluidas a partir de ellas, podemos inferir β:

$$\frac{\alpha \quad \alpha \rightarrow \beta}{\beta} \quad (\rightarrow E)$$

- **Eliminación de cuantificador universal.** Esta regla expresa que siempre que una fbf de la forma $\forall X\alpha$ pertenezca a las premisas o sea concluida a partir de ellas, una nueva fbf puede ser concluida al remplazar todas las ocurrencias libres de X en α por algún término t que es libre con respecto a X (todas las variables en t quedan libres al substituir X por t. La regla se presenta como sigue:

$$\frac{\forall X\alpha(X)}{\alpha(t)} \quad (\forall E)$$

- **Introducción de conjunción.** Cuando las fbf α y β pertenezcan a las premisas o sean concluidas a partir de ellas, podemos inferir $\alpha \wedge \beta$:

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta} \quad (\wedge I)$$

La correctez de estas reglas puede ser demostrada directamente a partir de la definición de la semántica de las fbf del lenguaje. El uso de las reglas de inferencia puede ilustrarse con el ejemplo formalizado. Las premisas son:

1. $\forall X \forall Y madre(X) \wedge hijoDe(Y, X) \rightarrow ama(X, Y)$
2. $madre(julia) \wedge hijoDe(luis, julia)$

Al aplicar la eliminación de cuantificador universal $(\forall X)$ a (1) obtenemos:

3. $\forall Y(madre(julia) \wedge hijoDe(Y, julia) \rightarrow ama(julia, Y)$

Al aplicar nuevamente $(\forall E)$ a (3) obtenemos:

4. $madre(julia) \wedge hijoDe(luis, julia) \rightarrow ama(julia, luis)$

Finalmente, al aplicar Modus Ponens a (2) y (4):

5. $ama(julia, luis)$

La conclusión (5) ha sido obtenida rigurosamente, aplicando las reglas de inferencia. Esto ilustra el concepto de **derivación**. El hecho de que una fórmula α sea derivable a partir de un conjunto de fórmulas Δ se escribe $\Delta \vdash \alpha$. Si las reglas de inferencia son sólidas (en el sentido técnico de *soundness*), siempre que $\Delta \vdash \alpha$ entonces $\Delta \models \alpha$. Esto es, si nuestra lógica es sólida, cualquier fbf que puede ser derivada de otra fbf, es tambien una consecuencia lógica de ésta última.

Derivación

Definición 2.7 (Solidez y completitud)**.** *Un conjunto de reglas de inferencia se dice* sólido *si, para todo conjunto de fbf cerradas (sin ocurrencia de variables libres) Δ y cada fbf cerrada α, siempre que $\Delta \vdash \alpha$ se tiene que $\Delta \models \alpha$. Las reglas de inferencia se dicen* completas *si $\Delta \vdash \alpha$ siempre que $\Delta \models \alpha$.*

En lo que sigue, explicaremos como automatizar el proceso de inferencia en un subconjunto de la Lógica de Primer Orden conocido como Programas Definitivos.

2.2 PROGRAMAS DEFINITIVOS

La idea central de la Programación Lógica es usar la computadora para obtener conclusiones a partir de descripciones declarativas, como las introducidas en la sección anterior. Estas descripciones, llamadas **programas lógicos**, consisten en un conjunto finito de fórmulas bien formadas (fbfs) de la lógica de Primer Orden. Esta idea tiene sus raíces en la demostración automática de teoremas, sin embargo, pasar de la demostración automática de teoremas experimental a la programación lógica aplicada, requiere mejoras con respecto a la eficiencia del sistema propuesto. Tales mejoras se logran imponiendo restricciones sobre las fbfs del lenguaje utilizado, de forma que podamos usar una poderosa regla de inferencia conocida como principio de **resolución-SLD**. Esta sección introduce el concepto de cláusula y programa lógico definitivos. El aparato técnico aquí presentado se basa principalmente en el texto de Nilsson y Maluszynski [62].

Programas lógicos

Resolución-SLD

2.2.1 Cláusulas definitivas

Consideremos una clase especial de enunciados declarativos del lenguaje natural, que utilizamos para describir hechos y reglas positivos. Un enunciado de este tipo puede especificar:

- Que una relación se mantiene entre elementos del universo de discurso (hechos).
- Que una relación se mantiene entre elementos del universo de discurso, si otras relaciones se mantienen (reglas).

Consideren los siguientes enunciados en lenguaje natural:

1. Antonio es hijo de Juan.
2. Ana es hija de Antonio.
3. Juan es hijo de Marcos.
4. Alicia es hija de Juan.
5. El nieto de una persona es el hijo del hijo de esa persona.

Estos enunciados pueden formalizarse en dos pasos. Primero, procedemos con las fbf atómicas que describen hechos:

1. hijo_de(antonio, juan)
2. hijo_de(ana,antonio)
3. hijo_de(juan,marcos)
4. hijo_de(alicia,juan)

El último enunciado puede aproximarse como: Para toda X e Y, X es nieto de Y si existe alguna Z tal que Z es hijo de Y y X es hijo de Z. En lógica de Primer Orden, esto se escribiría [3]:

$$\forall X \forall Y (nieto\_de(X, Y) \leftarrow \exists Z (hijo\_de(Z, Y) \wedge hijo\_de(X, Z)))$$

Usando las equivalencias de la lógica de Primer Orden (en particular $\alpha \rightarrow \beta \equiv \neg\alpha \vee \beta$; y la equivalencia entre cuantificadores $\forall X\alpha \equiv \neg\exists X\neg\alpha$), esta fbf puede escribirse de diversas maneras:

$$\forall X \forall Y (nieto\_de(X, Y) \vee \neg\exists Z (hijo\_de(Z, Y) \wedge hijo\_de(X, Z)))$$

$$\forall X \forall Y (nieto\_de(X, Y) \vee \forall Z \neg (hijo\_de(Z, Y) \wedge hijo\_de(X, Z)))$$

$$\forall X \forall Y \forall Z (nieto\_de(X, Y) \vee \neg (hijo\_de(Z, Y) \wedge hijo\_de(X, Z)))$$

$$\forall X \forall Y \forall Z (nieto\_de(X, Y) \leftarrow (hijo\_de(Z, Y) \wedge hijo\_de(X, Z)))$$

Observen que estas fbf están cerradas (no contienen variables fuera del alcance de los cuantificadores) bajo el cuantificador universal. Además, la regla tiene la siguiente estructura:

$$\alpha_0 \leftarrow \alpha_1 \wedge \cdots \wedge \alpha_n \quad (n \geq 0)$$

Los bloques de construcción α_i de estas fbf, se conocen como literales.

3 Observen que la implicación está invertida ($\leftarrow$) a la usanza de Prolog.

Definición 2.8 (Literal)**.** *Una literal es un átomo o la negación de un átomo. Una literal positiva es un átomo. Una literal negativa es la negación de un átomo.*

Un ejemplo de literal positiva es $hijo\_de(juan, marcos)$. Un ejemplo de literal negativa es $\neg hijo\_de(juan, alicia)$. Si p y q son predicados y f es un functor, entonces $p(X, alicia)$ y $q(Y)$ son literales positivas. $\neg q(alicia, f(Y))$ es una literal negativa.

Definición 2.9 (Cláusula)**.** *Una cláusula es una disyunción finita de cero o más literales.*

Definición 2.10 (Cláusula definitiva)**.** *Una cláusula se dice definitiva, si tiene exactamente una literal positiva.*

$$\alpha_0 \vee \neg\alpha_1 \vee \cdots \vee \neg\alpha_n \quad (n \geq 0)$$

lo cual es equivalente a la forma general de fbf que nos interesaba:

$$\alpha_0 \leftarrow \alpha_1 \wedge \cdots \wedge \alpha_n \quad (n \geq 0)$$

Si $n = 0$ tenemos por definición que la literal α_0 será una literal positiva, por lo que la cláusula definitiva toma la forma de un **hecho**. El cuerpo vacío puede representarse por el conectivo nulo ■, que es verdadero en toda interpretación (por simetría también se asume un conectivo nulo □, que es falso en toda interpretación). Si $n > 0$ la cláusula definitiva toma la forma de una **regla**, donde α_0 se conoce como **cabeza** de la regla; y la conjunción $\alpha_1 \wedge \cdots \wedge \alpha_n$ se conoce como **cuerpo** de la regla.

Hecho

Regla

El ejemplo de la relación $nieto\_de/2$ y la regla que lo define, muestra que las cláusulas definitivas usan una forma restringida de cuantificación existencial, las variables que ocurren sólo en el cuerpo de la cláusula están cuantificadas existencialmente en el cuerpo de la cláusula (el mismo ejemplo muestra que esto equivale a que tales variables estén cuantificadas universalmente sobre toda la fbf).

2.2.2 Programas y metas definitivos

La definición de programa definitivo es ahora directa:

Definición 2.11 (Programa definitivo)**.** *Un programa definitivo es un conjunto finito de cláusulas definitivas.*

Si una cláusula tiene sólo literales negativas, estamos hablando de una meta definitiva:

Definición 2.12 (Meta definitiva)**.** *Una cláusula sin literales positivas es una meta definitiva.*

$$\leftarrow \alpha_1 \wedge \cdots \wedge \alpha_n \quad (n \geq 1)$$

Definición 2.13 (Cláusula de Horn)**.** *Una cláusula de Horn es una cláusula definitiva ó una meta definitiva.*

Observen que a partir de estas definiciones, la cláusula vacía $\square$ [4] es una meta definitiva y, por lo tanto, una cláusula de Horn.

Adoptar a las cláusulas de Horn para abordar los programas y metas definitivos, constituye una restricción. Por ejemplo, no podemos expresar $p(a) \vee p(b)$. Esta perdida en expresividad se ve compensada por la ganancia en tratabilidad. Debido a su estructura restringida, las cláusulas de Horn son más fáciles de manipular que las cláusulas generales. En particular, esto es cierto para la deducción basada en resolución-SLD, que resulta completa para las cláusulas de Horn.

El significado lógico de las metas puede explicarse haciéndo referencia a la fbf equivalente cuantificada universalmente:

$$\forall X_1 \ldots X_m \neg(\alpha_1 \wedge \cdots \wedge \alpha_n)$$

donde las X_i son todas variables que ocurren en la meta. Esto es equivalente a:

$$\neg\exists X_1 \ldots X_m(\alpha_1 \wedge \cdots \wedge \alpha_n)$$

Esto puede verse como una pregunta existencial que el sistema tratará de negar, mediante la construcción de un contra ejemplo. Esto es, el sistema tratará de encontrar términos $t_i \ldots t_m$ tales que las fbf obtenidas a partir de $\alpha_1 \wedge \cdots \wedge \alpha_m$ al remplazar la variable X_i por t_i $(1 \leq i \leq m)$ son verdaderas en todo modelo del programa. Es decir, el sistema construirá una consecuencia lógica del programa que es un caso de la conjunción de todas las submetas de la meta.

Al dar una meta definitiva, el usuario selecciona un conjunto de conclusiones a ser construídas. Este conjunto puede ser finito o infinito. El problema de como construir tal conjunto lo veremos al tratar la resolución SLD.

Ejemplo 2.1. *Tomando en cuenta los hechos y reglas sobre una familia presentados al principio de esta sesión, el usuario podría estar interesado en las siguientes consultas (se muestra también la meta definitiva correspondiente):*

Consulta	***Meta definitiva***
¿Es Ana hija de Antonio?	$\leftarrow hijo(ana, antonio)$
¿Quién es nieto de Ana?	$\leftarrow nieto(X, ana)$
¿De quién es nieto Antonio?	$\leftarrow nieto(antonio, X)$
¿Quién es nieto de quién?	$\leftarrow nieto(X, Y)$

Las respuestas obtenidas serían:

- *Puesto que la primer meta no contiene variables, la respuesta sería Si (Yes).*
- *Puesto que el programa no contiene información sobre los nietos de Ana, la respueta a la segunda consulta es No (o ninguno).*
- *Puesto que Antonio es nieto de Marcos, la respuesta obtenida sería $X = marcos$.*

4 En realidad, la cláusula vacía tiene la forma $\square \leftarrow \blacksquare$ que equivale a $\square$ (Recuerden la tabla de verdad de la implicación). En algunos textos pueden encontrar esta expresión denotada por $\bot \leftarrow \top$.

- *La consulta final obtiene tres respuestas: $X = antonio$ $Y = alicia$, $X = alicia$ $Y = marcos$, $X = ana$ $Y = juan$.*

Es posible hacer consultas más elaboradas como ¿Hay alguna persona cuyos nietos son Antonio y Alicia?

$$\leftarrow nieto(antonio, X) \wedge nieto(alicia, X)$$

cuya respuesta esperada es $X = marcos$.

2.2.3 El modelo mínimo de Herbrand

Conocimiento positivo

Recuerden que los programas definitivos solo pueden expresar **conocimiento positivo**, tanto los hechos, como las reglas, nos dicen que elementos de una estructura están en una relación, pero no nos dicen cuales no. Por lo tanto, al usar el lenguaje de los programas definitivos, no es posible construir descripciones contradictorias, es decir, conjuntos de fbf no satisfacibles. En otras palabras, todo programa definitivo tiene un modelo. Veremos que todo programa definitivo tiene un **modelo mínimo** bien definido, que refleja toda la información expresada por el programa y nada más que eso. Recordemos que una interpretación que hace verdadera una fbf es su modelo.

Modelo mínimo

Existe una clase interesante de modelos, llamados de Herbrand en honor del francés Jacques Herbrand. En esta sección estudiaremos algunas propiedades de los modelos de Herbrand que explican porque son útiles y necesarios en el contexto de la Programación Lógica. Además, constataremos que los modelos de Herbrand proveen una semántica natural para los programas definitivos. Comenzaremos definiendo el **Universo y la Base de Herbrand**:

Universo y Base de Herbrand

Definición 2.14 (Universo y Base de Herbrand)**.** *Sea L un alfabeto que contiene al menos un símbolo de constante ($|Const| \geq 1$). El Universo de Herbrand U_L es el conjunto de todos los términos formados con las constantes y functores de L. La Base de Herbrand B_L es el conjunto de todos los átomos que pueden formarse con los predicados y los términos en el Universo de Herbrand U_L.*

Alfabeto

El universo y la base de Herbrand se definen normalmente para un programa dado. En ese caso, se asume que el **alfabeto** L consiste exactamente de aquellos símbolos que aparecen en el programa. Se asume también que el programa tiene al menos una constante (de otra forma el dominio estaría vacío).

Ejemplo 2.2. *Consideren el siguiente programa definitivo:*

$$\Delta = \{impar(s(0)), impar(s(s(X))) \leftarrow impar(X)\}$$

Si restringimos el lenguaje L a los símbolos que aparecen en este programa definitivo, tenemos que el universo de Herbrand es:

$$U_L = \{0, s(0), s(s(0)), s(s(s(0))), \dots\}$$

Puesto que el programa sólo incluye al predicado $impar$, la base de Herbrand se define como:

$$B_L = \{impar(0), impar(s(0)), impar(s(s(0))), \dots\}$$

Ejemplo 2.3. *Consideren este otro programa* $\Delta = \{p(a), q(a, f(b), q(X, X) \leftarrow p(X)\}$. *Sea L es lenguaje de Primer Orden dado por los símbolos en* Δ. *El Universo de Herbrand* U_L *es el conjunto infinito:*

$$U_L = \{a, b, f(a), f(b), f(f(a)), f(f(b)), \dots\}$$

Y la base de Herbrand es:

$$B_L = \{p(a), p(b), q(a, b), p(f(a)), p(f(b)), q(a, f(a)), q(a, f(b)), \dots\}$$

Lo que hace especial a una intrepretación de Herbrand es que se toma el conjunto de todos los términos sin variables (U_L) como el dominio de la interpretación; y la interpretación de todo término de base, es el término mismo.

Definición 2.15 (Interpretación de Herbrand)**.** *Una interpretación de Herbrand V de un programa definitivo* Δ, *es una interpretación donde:*

- *El dominio de V es* U_Δ.
- *Para cada constante* $c \in \Delta$, $c^V = c$.
- *Para cada functor* $f/n \in \Delta$, *se tiene un mapeo* f^V *de* U_L^n *a* U_L *definido por* f^V.
- *Para cada predicado* $p/n \in \Delta$, $p^V \subseteq U_\Delta^n$.

Definición 2.16 (Modelo de Herbrand)**.** *Un modelo de Herbrand de un conjunto de fbf (cerradas)* Δ *es una interpertación de Herbrand que es un modelo para todas las fbf en* Δ.

Observen que una interpretación de Herbrand V está completamente especificada por el conjunto de todas las $\alpha \in B_\Delta$ que son verdaderas bajo V. En otras palabras, una interpretación de Herbrand, es un subconjunto de la Base de Herbrand $\alpha \in B_\Delta | \models_V \alpha$.

Ejemplo 2.4. *Consideren el programa* Δ *en el ejemplo 2.2. Una posible interpretación de este programa es* $impar^V = \{s(0), s(s(s(0))), \dots\}$. *Una intepretación de Herbrand se puede especificar mediante una familia de tales relaciones (una por cada símbolo de predicado).*

Ejemplo 2.5. *Consideren ahora algunas interpretaciones de Herbrand de* Δ *tal y como se definió en el ejemplo 2.3:*

$$\begin{aligned} V_1 &= \{p(a), p(b), q(a, b,), q(b, b)\} \\ V_2 &= \{p(a), q(a, a), q(a, f(b))\} \\ V_3 &= \{p(f(f(a))), p(b), q(a, a), q(a, f(b))\} \\ V_4 &= \{p(a), p(b), q(a, a), q(b, b), q(a, f(b))\} \end{aligned}$$

V_2 y V_4 son modelos de Herbrand de $\Delta = \{p(a), q(a, f(b), q(X, X) \leftarrow p(X)\}$. *$V_1$ y V_3 no lo son.*

Resultados concernientes a los modelos de Herbrand

Las interpretaciones y los modelos de Herbrand tienen dos propiedades atractivas. La primera es pragmática: Para determinar si una interpretación de Herbrand V es un modelo de una fbf cuantificada universalmente $\forall\alpha$, es suficiente verificar si α es verdadera en V, para todas las asignaciones posibles de las variables de α.

La segunda razón para considerar las interpretaciones de Herbrand es más teórica: Para el lenguaje restringido de cláusulas definitivas, si queremos verificar que una fbf atómica α es consecuencia de un programa definitivo Δ basta con verificar que todo modelo de Herbrand de Δ es también un modelo de Herbrand de α.

Teorema 2.1. *Sea Δ un programa definitivo y γ una meta definitiva. Si V' es un modelo de $\Delta \cup \{\gamma\}$, entonces $V = \{\alpha \in B_\Delta | \models_{V'} \alpha\}$ es un modelo de Herbrand de $\Delta \cup \{\gamma\}$.*

Claramente V es una interpretación de Herbrand, bajo las definiciones precedentes. Ahora, asumanos que V no es un modelo de $\Delta \cup \{\gamma\}$. Es decir, existe un caso de base de una cláusula en $\Delta \cup \{\gamma\}$, que no es verdadera en V. Puesto que la cláusula es falsa, su cuerpo es verdadero y su cabeza falsa en V'. Pero V' es un modelo de $\Delta \cup \{\gamma\}$, por lo que, por contradicción V es un modelo $\Delta \cup \{\gamma\}$.

Ahora bien, observen que este teorema es válido sólo para conjuntos de cláusulas definitivas. En el caso general, la no existencia del modelo de Herbrand es insuficiente para probar que Δ es no satisfacibe. Por ejemplo, consideren a L como un lenguaje de primer orden formado por los símbolos en $\Delta = \{\exists X p(X), \neg p(a)\}$. Δ tiene un modelo, pero este no es un modelo de Herbrand (Una interpretación posible es aquella cuyo significado pretendido es que o no es impar y existe un número natural que lo es, p.ej. 1).

Es común que a partir de un conjunto Δ y una fbf α, queremos encontrar si $\Delta \models \alpha$. Esto es cierto si cada modelo de Δ es también un modelo de α. Lo interesante viene ahora:

Proposición 2.1. *Sea Δ un programa definitivo y α una cláusula definitiva. Sea $\Delta' = \Delta \cup \{\neg\alpha\}$. Entonces $\Delta \models \alpha$ si y sólo si Δ' no tiene modelo de Herbrand.*

La prueba de esta proposición es como sigue: $\Delta \models \alpha$ si y sólo si $\Delta \cup \{\neg\alpha\}$ no es satisfacible. Esto es cierto, si y sólo si Δ no tiene modelos y por lo tanto, no tiene modelo de Herbrand.

Lo que esta proposición nos dice es que si queremos probar que $\Delta \models \alpha$, sólo debemos considerar modelos de Herbrand de la forma V. Aunque el número de interpretaciones de Herbrand es normalmente infinito, la tarea de investigar interpretaciones de Herbrand es más tratable que la de investigar cualquier interpretación arbitraria, puesto que nos restringimos a un dominio único definido por el Universo de Herbrand U_Δ.

Observen que la base de Herbrand de un programa definitivo Δ es siempre un modelo de Herbrand del programa. Sin embargo, es un modelo nada interesante, esto es, cada predicado n-ario en el programa es interpretado como la relación n-aria completa sobre el dominio de los terminos cerrados. ¿Qué es lo que hace a un modelo de programa interesante? En lo que sigue

Modelo mínimo de Herbrand

demostraremos la existencia de un modelo mínimo único, llamado el **modelo mínimo de Herbrand** de un programa definitivo. Luego mostraremos que este modelo contiene toda la información positiva presente en el programa.

Los modelos de Herbrand de un programa definitivo son subconjuntos de su base de Herbrand. Por lo tanto, la inclusión en conjuntos establece un orden natural en tales modelos. Para poder demostrar la existencia de modelos mínimos con respecto a la inclusión es suficiente demostrar que la intersección de todos los modelos de Herbrand es también un modelo de Herbrand.

Teorema 2.2 (Intersección de modelos)**.** *Sea M una familia no vacía de modelos de Herbrand de un programa definitivo Δ. Entonces la intersección $V = \bigcap M$ es un modelo de Herbrand de Δ.*

La demostración es como sigue: Supongamos que V no es un modelo de Δ. Por lo tanto existe una una cláusula de base en Δ, de la forma:

$$\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_n \quad (n \geq 0)$$

que no es verdera en V. Esto ímplica que V contiene a $\alpha_1, \ldots \alpha_n$, pero no a α_0. Luego, $\alpha_1, \ldots, \alpha_n$ son miembros de toda interpretación en la familia M. Más importante aún, debe existir un modelo $V_i \in M$ tal que $\alpha_0 \notin V_i$, de forma que la cláusula $\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_n (n \geq 0)$ no es verdadera en ese V_i. Por lo tanto V_i no es un modelo del programa Δ, lo que contradice nuestro supuesto.

Al tomar la intersección de todos los modelos de Herbrand (se sabe que todo programa definitivo tiene un modelo de Herbrand B_L) de un programa definitivo, obtenemos el modelo mínimo de Herbrand el programa.

Ejemplo 2.6. *Sea Δ el programa definitivo $\{masculino(adan), femenino(eva)\}$ con su interpretación obvia. Δ tiene los siguientes modelos de Herbrand:*

- $\{masculino(adan), femenino(eva)\}$
- $\{masculino(adan), masculino(eva), femenino(eva)\}$
- $\{masculino(adan), masculino(eva), femenino(adan)\}$
- $\{masculino(adan), masculino(eva), femenino(eva), femenino(adan)\}$

No es complicado confirmar que la intersección de estos modelos produce un modelo de Herbrand. Coincidentemente, todos los modelos salvo el primero, contienen átomos incompatibles con el significado esperado del programa. Este ejemplo nos muestra la conexión entre los modelos mínimos de Herbrand y el modelo intentado de un programa definitivo. Este modelo es una abstracción del mundo a ser descrita por el programa. El mundo puede ser más rico que el modelo mínimo de Herbrand. Por ejemplo hay más $femenino$s que *eva*. Sin embargo, aquella información que no se provea explícitamente (hechos) o implícitamente (reglas) no puede ser obtenida como respuesta a una meta. Las respuestas corresponden a las consecuencias lógicas del programa.

Teorema 2.3. *El modelo mínimo de Herbrand M_Δ de un programa definitivo Δ es el conjunto de todas las consecuencias lógicas atómicas de base del programa. Esto es: $M_\Delta = \{\alpha \in B_\Delta | \Delta \models \alpha\}$.*

La prueba de este teorema pasa por demostrar que $M_\Delta \supseteq \{\alpha \in B_L | \Delta \models \alpha\}$ y que $M_\Delta \subseteq \{\alpha \in B_\Delta | \Delta \models \alpha\}$.

Construcción del modelo mínimo de Herbrand

¿Cómo podemos construir el modelo mínimo de Herbrand? o ¿Cómo puede aproximarse sucesivamente por medio de la enumeración de sus elementos? La respuesta a estas preguntas se da mediante una aproximación de punto fijo [5] a la semántica de los programas definitivos.

Un programa definitivo está compuesto de hechos y reglas. Es evidente que todos los hechos deben incluirse en cualquier modelo de Herbrand. Si la interpretación V no incluye el hecho α del programa Δ, entonces V no es un modelo de Herbrand de Δ.

Ahora consideremos una regla de la forma $\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_n (n > 0)$. La regla especifica que siempre que $\alpha_1, \ldots, \alpha_n$ son verdaderas, también lo es α_0. Esto es, tomando cualquier asignacion de valores θ que haga que la regla no tenga variables sin valor $(\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_n)\theta$. Si la interpretación V incluye a $\alpha_1\theta, \ldots \alpha_n\theta$, deberá incluir también a $\alpha_0\theta$ para ser un modelo.

Consideren ahora el conjunto V_1 de todos los hechos sin variables de el programa. Es posible utilizar cada regla para aumentar V_1 con nuevos elementos que necesariamente pertenencen a todo modelo. De modo que se obtiene un nuevo conjunto V_2 que puede usarse para generar más elementos que pertenecen a todo modelo. El proceso se repite mientras puedan generarse nuevos elementos. Los elementos agregados a V_{i+1} son aquellos que se siguen inmediatamente de V_i.

La construcción así obtenida puede formalizarse como la iteración de una transformación T_Δ sobre las interpretaciones de Herbrand de un programa Δ. La operación se llama **operador de consecuencia inmediata** y se define como sigue:

Operador de consecuencia inmediata

Definición 2.17 (Operador de consecuencia inmediata)**.** *Sea base(Δ) el conjunto de todas las cláusulas de base en Δ. T_Δ es una función sobre las interpretaciones de Herbrand de Δ definida como sigue:*

$$T_\Delta(V) = \{\alpha_0 \mid \alpha_0 \leftarrow \alpha_1, \ldots, \alpha_n \in ground(\Delta) \wedge \{\alpha_1, \ldots, \alpha_n\} \subseteq V\}$$

Para los programas definitivos, se puede mostrar que existe una interpretación mínima V tal que $T_\Delta(V) = V$ y que V es idéntica al modelo mínimo de Herbrand de Δ. Más aún, el modelo mínimo de Herbrand es el limite de la creciente, posiblemente infinita, secuencia de iteraciones:

$$\emptyset, T_\Delta(\emptyset), T_\Delta(T_\Delta(\emptyset)), \ldots$$

Existe una notación estándar para denotar a los miembros de esta secuencia de interpretaciones construídas a partir de Δ:

$$\begin{aligned} T_\Delta \uparrow 0 &= \emptyset \\ T_\Delta \uparrow (i+1) &= T_\Delta(T_\Delta \uparrow i) \\ T_\Delta \uparrow n &= \bigcup_{i=0}^{n} T_\Delta \uparrow i \end{aligned}$$

5 El punto fijo de una función $f : D \to D$ es un elemento $x \in D$ tal que $f(x) = x$.

Ejemplo 2.7. *Tomando Δ como el programa de impar (ej. 2.2, tenemos:*

$$\begin{aligned} T_\Delta \uparrow 0 &= \emptyset \\ T_\Delta \uparrow 1 &= \{impar(s(0))\} \\ T_\Delta \uparrow 2 &= \{impar(s(0)), impar(s(s(s(0))))\} \\ &\vdots \\ T_\Delta \uparrow m &= \{impar(s^n(0)) \mid n \in \{1,3,5,\dots\}\} \end{aligned}$$

Como mencionamos, el conjunto construído de esta manera es identico al modelo mínimo de Herbrand de Δ.

Teorema 2.4. *Sea Δ un programa definitivo y V_Δ su modelo mínimo de Herbrand. Entonces:*

- *V_Δ es la interpretación mínima de Herbrand tal que $T_\Delta(V_\Delta) = V_\Delta$.*
- *$V_\Delta = T_\Delta \uparrow n$.*

2.3 PRINCIPIO DE RESOLUCIÓN

Esta sección introduce un procedimiento de prueba utilizado ampliamente en la programación lógica: El principio de resolución propuesto por Robinson [74]. Si bien este procedimiento está orientado a un lenguaje más expresivo, nosotros nos concentraremos en una versión del principio que aplica a programas definitivos y se conoce como **resolución-SLD** [48] (Resolución lineal con función de selección para cláusulas definitivas). De esta forma, definiremos la teoría de prueba para los programas definitivos, cuyo lenguaje y teoría de modelo fueron definidos en las secciones previas. Para ello, recordaremos los conceptos de sistema lógico robusto y completo e introduciremos el concepto de consecuencia lógica decidible. Después ejemplificaremos un proceso de prueba sobre los programas lógicos definitivos. Posteriormente se abordan los conceptos técnicos de substitución y unificación, necesarios para definir la resolución-SLD. Finalmente se analizan algunas propiedades de este principio.

Resolución-SLD

2.3.1 Robustez y completitud

La programación lógica, en este caso restringida a programas definitivos, concierne el uso de la lógica para **representar** y resolver problemas. Este uso es ampliamente aceptado en la IA, donde la idea se resume como sigue: Un problema o sujeto de investigación puede describirse mediante un conjunto de cláusulas. Si tal descripción es lo suficientemente precisa, la solución al problema o la respuesta a la pregunta planteada en la investigación, es una consecuencia lógica del conjunto de cláusulas que describen el problema. De forma que nos gustaría tener un procedimiento algorítmico, que permita establecer si $\Delta \models \phi$ es el caso, esto es, si ϕ es consecuencia lógica de Δ.

Representación de conocimiento

En el caso de la lógica proposicional, el número de interpretaciones posibles para $\Delta \cup \{\phi\}$ es finito, de hecho es 2^n donde n es el número de átomos distintos que ocurren en $\Delta \cup \{\phi\}$. Para computar si $\Delta \models \phi$ simplemente debemos buscar si los modelos de Δ, lo son también de ϕ. Por tanto, se dice que la consecuencia en la lógica proposicional es **decidible**, es decir, existe un algoritmo que puede resolver el problema. Pero ¿Qué sucede en el contexto de la lógica de Primer Orden?

Procedimiento de prueba decidible

La intuición nos dice que el procedimiento de decisión de la lógica proposicional no es adecuado para las representaciones en primer orden, pues en este caso podemos tener una cantidad infinita de interpretaciones diferentes. Lo que es peor, el teorema de Church [13, 88], muestra que la lógica de Primer Orden es **no decidible** en lo general:

Procedimiento de prueba no decidible

Teorema 2.5 (Church)**.** *El problema de si $\Delta \models \phi$, cuando Δ es un conjunto finito arbitrario de fbf, y ϕ es una fbf arbitraria, es **no decidible**.*

Observen que el problema es no decidible para fórmulas de Primer Orden arbitrarias. No existe un algoritmo que en un número finito de pasos, de la respuesta correcta a la pregunta ¿Es ϕ consecuencia lógica de Δ? para el caso general. Afortunadamente existen procedimientos de prueba que pueden responder a esta cuestión en un número finito de pasos, cuando es el caso que $\Delta \models \phi$. Pero si es el caso que $\Delta \not\models \phi$ obtendremos la respuesta "no" (en el mejor de los casos) o el procedimiento no terminará nunca. Por ello suele decirse que la consecuencia en estos casos es **semi-decidible**.

Procedimiento de prueba semi-decidible

Hemos abordado el concepto de **procedimiento de prueba**, al introducir el concepto de consecuencia válida y probar que el sistema es robusto y consistente. Recuerden que una consecuencia válida es aquella donde su conclusión ϕ es derivable a partir de sus premisas Δ, mediante el uso de las reglas de inferencia del sistema. Tal derivación, o prueba, suele consistir de un pequeño número de transformaciones, en las cuales nuevas fbf son derivadas de las premisas y de fbf previamente derivadas. Para el caso proposicional, si ϕ es derivable de Δ entonces es el caso que $\Delta \models \phi$ (Robustez); y si ese es el caso, entonces existe una prueba de ello (Completez).

Procedimiento de prueba

Ahora bien, con nuestro conocido *modus ponens*:

$$\frac{\phi, \quad \phi \rightarrow \psi}{\psi}$$

y el siguiente programa lógico:

$$\Delta = \{p(a), q(b) \leftarrow p(a), r(b) \leftarrow q(b)\}$$

es posible construir la prueba de $r(b)$ como sigue:

1. Derivar $q(b)$ a partir de $p(a)$ y $q(b) \leftarrow p(a)$.
2. Derivar $r(b)$ a partir de $q(b)$ y $r(b) \leftarrow q(b)$.

Es evidente que si usamos *modus ponens*, la conclusión ψ es una consecuencia lógica de las premisas: $\{\phi, \phi \rightarrow \psi\} \models \psi$. A esta propiedad del *modus ponens* se le conoce como robustez (*soundness*). En general un procedimiento de prueba es **robusto** si todas las fbf ψ que pueden ser derivadas de algún

Robustez

conjunto de fbfs Δ usando el procedimiento, son consecuencias lógicas de Δ. En otras palabras, un procedimiento de prueba es robusto si y sólo si sólo permite derivar consecuencias lógicas de las premisas.

Una segunda propiedad deseable de los procedimientos de prueba es su completez. Un procedimiento de prueba es **completo** si toda fbf que es una consecuencia lógica de las premisas Δ, puede ser derivada usando el procedimiento en cuestión. El *modus ponens* por si mismo, no es completo. Por ejemplo, no existe secuencia alguna de aplicaciones del *modus ponens* que deriven la fbf $p(a)$ de $\Delta = \{p(a) \wedge p(b)\}$, cuando es evidente que $\Delta \models p(a)$.

Completez

La regla $\frac{\phi}{\psi}$ es completa, pero no robusta. !Nos permite extraer cualquier conclusión, a partir de cualquier premisa! Esto ejemplifica que obtener completez es sencillo, pero obtener completez y robustez, no lo es.

2.3.2 Pruebas y programas lógicos

Recordemos que las cláusulas definitivas tienen la estructura general de la implicación lógica:

$$\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_n \quad (n \geq 0)$$

donde $\alpha_0, \ldots, \alpha_n$ son átomos. Consideren el siguiente programa definitivo Δ que describe un mundo donde los padres de un recién nacido están orgullosos, Juan es el padre de Ana y Ana es una recién nacida:

$$\begin{aligned} orgulloso(X) &\leftarrow padre(X,Y), recien\_nacido(Y). \\ padre(X,Y) &\leftarrow papa(X,Y). \\ padre(X,Y) &\leftarrow mama(X,Y). \\ papa(juan, ana). & \\ recien\_nacido(ana). & \end{aligned}$$

Observen que el programa describe únicamente conocimiento positivo, es decir, no especifica quién no está orgulloso. Tampoco qué significa para alguien no ser padre. Ahora, supongamos que deseamos contestar la pregunta ¿Quién está orgulloso? Esta pregunta concierne al mundo descrito por nuestro programa, esto es, concierne al modelo pretendido de Δ. La respuesta que esperamos es, por supuesto, *juan*. Ahora, recuerden que la lógica de Primer Orden no nos permite expresar enunciados interrogativos, por lo que nuestra pregunta debe formalizarse como una cláusula meta (enunciado declarativo):

$$\leftarrow orgulloso(Z).$$

que es una abreviatura de $\forall Z \neg orgulloso(Z)$ (una cláusula definitiva sin cabeza), que a su vez es equivalente de:

$$\neg \exists Z \; orgulloso(Z).$$

cuya lectura es –Nadie está orgulloso; esto es, la respuesta negativa a la consulta original –¿Quién está orgulloso? La meta ahora es probar que este

enunciado es falso en todo modelo del programa Δ y en particular, es falso en el modelo previsto para Δ, puesto que esto es una forma de probar que $\Delta \models \exists Z\ orgulloso(Z)$. En general para todo conjunto de fbf cerradas Δ y una fbf cerrada γ, tenemos que $\Delta \models \gamma$ si $\Delta \cup \{\neg\gamma\}$ es no satisfacerle (no tiene modelo). Por lo tanto, nuestro objetivo es encontrar una substitución θ tal que el conjunto $\Delta \cup \{\neg orgulloso(Z)\theta\}$ sea no satisfacible, o de manera equivalente, $\Delta \models \exists Z\ orgulloso(Z)\theta$.

El punto inicial de nuestro razonamiento es asumir la meta G_0 – Para cualquier Z, Z no está orgulloso. La inspección del programa Δ revela que una regla describe una condición para que alguien esté orgulloso:

$$orgulloso(X) \leftarrow padre(X, Y), recien\_nacido(Y).$$

lo cual es lógicamente equivalente a:

$$\forall(\neg orgulloso(X) \rightarrow \neg(padre(X, Y) \wedge recien\_nacido(Y)))$$

Al renombrar X por Z, eliminar el cuantificador universal y usar *modus ponens* con respecto a G_0, obtenemos:

$$\neg(padre(Z, Y) \wedge recien\_nacido(Y))$$

o su equivalente:

$$\leftarrow padre(Z, Y), recien\_nacido(Y).$$

al que identificaremos como G_1. Un paso en nuestro razonamiento resulta en remplazar la meta G_0 por la meta G_1 que es verdadera en todo modelo $\Delta \cup \{G_0\}$. Ahora solo queda probar que $\Delta \cup \{G_1\}$ es no satisfacible. Observen que G_1 es equivalente a la fbf:

$$\forall Z \forall Y(\neg padre(Z, Y) \vee \neg recien\_nacido(Y))$$

Por lo tanto, puede probarse que la meta G_1 es no satisfacible para Δ, si en todo modelo de Δ hay una persona que es padre de un recién nacido. Entonces, verificamos primero si hay padres con estas condiciones. El programa contiene la cláusula:

$$padre(X, Y) \leftarrow papa(X, Y).$$

que es equivalente a:

$$\forall(\neg padre(X, Y) \rightarrow \neg papa(X, Y))$$

por lo que G_1 se reduce a:

$$\leftarrow papa(Z, Y), recien\_nacido(Y).$$

que identificaremos como G_2. Se puede mostrar que no es posible satisfacer la nueva meta G_2 con el programa Δ, si en todo modelo de Δ hay una persona que es papá de un recién nacido. El programa declara que *juan* es padre de *ana*:

$$papa(juan, ana).$$

así que sólo resta probar que "*ana* no es una recién nacida" no se puede satisfacer junto con Δ:

$$\leftarrow recien\_nacido(ana).$$

pero el programa contiene el hecho:

$$recien\_nacido(ana).$$

equivalente a $\neg recien\_nacido(ana) \rightarrow \square$, lo que conduce a una refutación. Este razonamiento puede resumirse de la siguiente manera: para probar la existencia de algo, suponer lo contrario y usar *modus ponens* y la regla de eliminación del cuantificador universal, para encontrar un contra ejemplo al supuesto.

Observen que la meta definitiva fue convertida en un conjunto de átomos a ser probados. Para ello, se seleccionó una fbf atómica de la meta $\alpha_0(\sigma_1, \ldots, \sigma_n)$ y una cláusula de la forma $\alpha_0(\tau_1, \ldots, \tau_n) \leftarrow \alpha_1, \ldots \alpha_n$ para encontrar una instancia común de la submeta y la cabeza de la cláusula seleccionada; es decir, una substitución θ ambas expresiones sean idénticas. Tal substitución se conoce como **unificador**. La nueva meta se construye reemplazando el átomo seleccionado en la meta original, por el cuerpo de la cláusula seleccionada, bajo la substitución θ.

Unificador

El paso de computación básico de nuestro ejemplo, puede verse como una regla de inferencia puesto que transforma fórmulas lógicas. Lo llamaremos **principio de resolución SLD** para programas definitivos. Como mencionamos, el procedimiento combina *modus ponens*, *eliminación del cuantificador universal* y en el paso final un *reductio ad absurdum*.

Cada paso de razonamiento produce una substitución, si se prueba en k pasos que la meta definida en cuestión no puede satisfacerse, probamos que:

$$\leftarrow (\alpha_1, \ldots \alpha_n)\theta_1 \ldots \theta_k$$

es una instancia que no puede satisfacerse. De manera equivalente, que:

$$\Delta \models (\alpha_1 \wedge \cdots \wedge \alpha_n)\theta_1 \ldots \theta_k$$

Observen que generalmente, la computación de estos pasos de razonamiento no es determinista: cualquier átomo de la meta puede ser seleccionado y pueden haber varias cláusulas del programa que unifiquen con el átomo seleccionado. Otra fuente de indeterminismo es la existencia de unificadores alternativos para dos átomos. Esto sugiere que es posible construir muchas soluciones (algunas veces, una cantidad infinita de ellas).

Por otra parte, es posible también que el atomo seleccionado no unifique con ninguna cláusula en el programa. Esto indica que no es posible construir un contra ejemplo para la meta definida inicial. Finalmente, la computación puede caer en un ciclo y de esta manera no producir solución alguna.

2.3.3 Substitución

Recordemos que una substitución es un conjunto de asignaciones de términos a variables. Por ejemplo, podemos remplazar la variable X por el

término $f(a)$ en la cláusula $p(X) \vee q(X)$, y así obtener la nueva cláusula $p(f(a)) \vee q(f(a))$. Si asumimos que las cláusulas están cuantificadas universalmente, decimos que está substitución hace a la cláusula original *menos general* ó *más específica*, en el sentido que mientras la cláusula original dice que para toda X la cláusula se satisface, la segunda versión es sólo cierta cuando X es substituida por a. Observen que la segunda cláusula es consecuencia lógia de la primera: $p(X) \vee q(X) \models p(f(a)) \vee q(f(a))$

Definición 2.18 (Substitución)**.** *Una substitución θ es un conjunto finito de la forma:*

$$\{X_1/t_1, \ldots, X_n/t_n\}, \quad (n \geq 0)$$

donde las X_i son variables, distintas entre si, y los t_i son términos. Decimos que t_i substituye a X_i. La forma X_i/t_i se conoce como ligadura de X_i.

Substitución de base

Una substitución se dice **de base** si todos los términos en ella son de base. Es claro, que éste es el caso de las interpretaciones de Herband.

Substitución identidad

La substitución definida como el conjunto vacío, se conoce como **substitución de identidad** o **substitución vacía** y se denota por ϵ. La restricción de θ sobre un conjunto de variables *Var* es la substitución $\{X/t \in \theta \mid X \in Var\}$.

Ejemplo 2.8. *$\{Y/X, X/g(X,Y)\}$ y $\{X/a, Y/f(Z), Z/(f(a), X_1/b\}$ son substituciones. La restricción de la segunda substitución sobre $\{X, Z\}$ es $\{X/a, Z/f(a)\}$.*

Definición 2.19 (Expresión)**.** *Una expresión es un término, una literal, o una conjunción o disyunción de literales. Una expresión simple es un término o una literal.*

Observen que una cláusula es una expresión. Las substituciones pueden aplicarse a las expresiones, lo que significa que las variables en las expresiones serán remplazadas de acuerdo a la substitución.

Definición 2.20. *Sea $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ una substitución y α una expresión. Entonces $\alpha\theta$, la ocurrencia (*instance*) de α por θ, es la expresión obtenida al substituir simultáneamente X_i por t_i para $1 \leq i \leq n$. Si $\alpha\theta$ es una expresión de base, se dice que es una ocurrencia base y se dice que θ es una substitución de base para α. Si $\Sigma = \{\alpha_1, \ldots, \alpha_n\}$ es un conjunto finito de expresiones, entonces $\Sigma\theta$ denota $\{\alpha_1\theta, \ldots, \alpha_n\theta\}$.*

Ejemplo 2.9. *Sea α la expresión $p(Y, f(X))$ y sea θ la substitución $\{X/a, Y/g(g(X))\}$. La ocurrencia de α por θ es $\alpha\theta = p(g(g(X)), f(a)$. Observen que X e Y son simultáneamente remplazados por sus respectivos términos, lo que implica que X en $g(g(X))$ no es afectada por X/a.*

Si α es una expresión cerrada que no es un término, por ejemplo, una literal, o una conjunción o disyunción de literales, y θ es una substitución, lo siguiente se cumple:

$$\alpha \models \alpha\theta$$

por ejemplo: $p(X) \vee \neg q(Y) \models p(a) \vee \neg q(Y)$ donde hemos usado la substitución $\{X/a\}$.

Podemos aplicar una substitución θ y luego aplicar una substitución σ, a lo cual se llama composición de las substituciones θ y σ. Si ese es el caso, primero se aplica θ y luego σ. Las composiciones pueden verse como mapeos del conjunto de variables en el lenguaje, al conjunto de términos.

Definición 2.21 (Composición). *Sean $\theta = \{X_1/s_1, \ldots, X_m/s_m\}$ y $\sigma = \{Y_1/t_1, \ldots Y_n/t_n\}$ dos substituciones. Consideren la secuencia:*

$$X_1/(s_1\sigma), \ldots, X_m/(s_m\sigma), Y_1/t_1, \ldots, Y_n/t_n$$

Si se borran de esta sencuencia las ligaduras $X_i/s_i\sigma$ cuando $X_i = s_i\sigma$ y cualquier ligadura Y_j/t_j donde $Y_j \in \{X_1, \ldots, X_m\}$. La substitución consistente en las ligaduras de la secuencia resultante es llamada composición de θ y σ, se denota por $\theta\sigma$.

Ejemplo 2.10. *Sea $\theta = \{X/f(Y), Z/U\}$ y $\sigma = \{Y/b, U/Z\}$. Construimos la secuencia de ligaduras $X/(f(Y)\sigma), Z/(u)\sigma, Y/b, U/Z$ lo cual es $X/f(b), Z/Z, Y/b, U/Z$. Al borrar la ligadura Z/Z obtenemos la secuencia $X/f(b), Y/b, U/Z = \theta\sigma$.*

Definición 2.22 (Ocurrencia). *Sean θ y σ dos substituciones. Se dice que θ es una ocurrencia de σ, si existe una substitución γ, tal que $\sigma\gamma = \theta$.*

Ejemplo 2.11. *La substitución $\theta = \{X/f(b), Y/a\}$ es una ocurrencia de la substitución $\sigma = \{X/f(X), Y/a\}$, puesto que $\sigma\{X/b\} = \theta$.*

Algunas propiedades sobre las substituciones incluyen:

Proposición 2.2. *Sea α una expresión, y sea θ, σ y γ substituciones. Las siguientes relaciones se cumplen:*

1. $\theta = \theta\epsilon = \epsilon\theta$
2. $(\alpha\theta)\sigma = \alpha(\theta\sigma)$
3. $\theta\sigma)\gamma = \theta(\sigma\gamma)$

2.3.4 Unificación

Uno de los pasos principales en el ejemplo de la sección 2.3.2, consistió en hacer que dos fbf atómicas se vuelvan sintácticamente equivalentes. Este proceso se conoce como **unificación** y posee una solución algorítmica. *Unificación*

Definición 2.23 (Unificador). *Sean α y β términos. Una substitución θ tal que α y β sean idénticos ($\alpha\theta = \beta\theta$) es llamada unificador de α y β.*

Ejemplo 2.12.

$$unifica(conoce(juan, X), conoce(juan, maria)) = \{X/maria\}$$

$$unifica(conoce(juan, X), conoce(Y, Z)) = \{Y/juan, X/Z\}$$

$$= \{Y/juan, X/Z, W/pedro\}$$

$$= \{Y/juan, X/juan, Z/juan\}$$

Definición 2.24 (Generalidad entre substituciones)**.** *Una substitución θ se dice más general que una substitución σ, si y sólo si existe una substitución γ tal que $\sigma = \theta\gamma$.*

Definición 2.25 (MGU)**.** *Un unificador θ se dice el unificador más general (MGU) de dos términos, si y sólo si θ es más general que cualquier otro unificador entre esos términos.*

Ejemplo 2.13. *Consideren los términos $f(X)$ y $f(g(Y))$, el MGU de ellos es $\{X/g(Y)\}$, pero existen otros muchos unificadores no MGU, por ejemplo $\{X/g(a), Y/a\}$. Intuitivamente, el MGU de dos términos es el más simple de todos sus unificadores.*

Definición 2.26 (Forma resuelta)**.** *Un conjunto de ecuaciones $\{X_1 = t_1, \ldots, X_n = t_n\}$ está en forma resuelta, si y sólo si $X_1, \ldots, X_n$ son variables distintas que no ocurren en $t_1, \ldots, t_n$.*

Existe una relación cercana entre un conjunto de ecuaciones en forma resuelta y el unificador más general de ese conjunto: Sea $\{X_1 = t_1, \ldots, X_n = t_n\}$ un conjunto de ecuaciones en forma resuelta. Entonces $\{X_1/t_1, \ldots, X_n/t_n\}$ es un MGU idempotente de la forma resuelta.

Definición 2.27 (Equivalencia en conjuntos de ecuaciones)**.** *Dos conjuntos de ecuaciones E_1 y E_2 se dicen equivalentes, si tienen el mismo conjunto de unificadores.*

La definición puede usarse como sigue: para computar el MGU de dos términos α y β, primero intente transformar la ecuación $\{\alpha = \beta\}$ en una forma resuelta equivalente. Si esto falla, entonces $mgu(\alpha, \beta) = fallo$. Sin embargo, si una forma resuelta $\{X_1 = t_1, \ldots, X_n = t_n\}$ existe, entonces $mgu(\alpha, \beta) = \{X_1/t_1, \ldots, X_n/t_n\}$. Un algoritmo para encontrar la forma resuelta de un conjunto de ecuaciones es como sigue:

Ejemplo 2.14. *El conjunto $\{f(X, g(Y)) = f(g(Z), Z)\}$ tiene una forma resuelta, puesto que:*

$$\Rightarrow \{X = g(Z), g(Y) = Z\}$$
$$\Rightarrow \{X = g(Z), Z = g(Y)\}$$
$$\Rightarrow \{X = g(g(Y)), Z = g(Y)\}$$

Ejemplo 2.15. *El conjunto $\{f(X, g(X), b) = f(a, g(Z), Z)\}$ no tiene forma resuelta, puesto que:*

$$\Rightarrow \{X = a, g(X) = g(Z), b = Z\}$$
$$\Rightarrow \{X = a, g(a) = g(Z), b = Z\}$$
$$\Rightarrow \{X = a, a = Z, b = Z\}$$
$$\Rightarrow \{X = a, Z = a, b = Z\}$$
$$\Rightarrow \{X = a, Z = a, b = a\}$$
$$\Rightarrow fallo$$

Algoritmo 2.1 Unifica(E)

```
1:  function UNIFICA(E)                          ▷ E es un conjunto de ecuaciones
2:      repeat
3:          (s = t) ← seleccionar(E)
4:          if f(s_1,...,s_n) = f(t_1,...,t_n) (n ≥ 0) then
5:              remplazar (s = t) por s_1 = t_1,...,s_n = t_n
6:          else if f(s_1,...,s_m) = g(t_1,...,t_n) (f/m ≠ g/n) then
7:              return(fallo)
8:          else if X = X then
9:              remover la X = X
10:         else if t = X then
11:             remplazar t = X por X = t
12:         else if X = t then
13:             if subtermino(X,t) then
14:                 return(fallo)
15:             else remplazar todo X por t
16:             end if
17:         end if
18:     until No hay accion posible para E
19: end function
```

Ejemplo 2.16. *El conjunto* $\{f(X, g(X)) = f(Z, Z)\}$ *no tiene forma resuelta, puesto que:*

$$\Rightarrow \{X = Z, g(X) = Z\}$$
$$\Rightarrow \{X = Z, g(Z) = Z\}$$
$$\Rightarrow \{X = Z, Z = g(Z)\}$$
$$\Rightarrow fallo$$

Verificación de ocurrencia

Este algoritmo termina y regresa una forma resuelta equivalente al conjunto de ecuaciones de su entrada; o bien regresa fallo si la forma resuelta no existe. Sin embargo, el computar $subtermino(X, t)$ hace que el algoritmo sea altamente ineficiente. Los sistemas Prolog resuelven este problema haciéndo caso omiso de la verificación de ocurrencia. El standard ISO Prolog (1995) declara que el resultado de la unificación es no decidible. Al eliminar la verificación de ocurrencia es posible que al intentar resolver $X = f(X)$ obtengamos $X = f(f(X)) \cdots = f(f(f \ldots))$. En la practica los sistemas Prolog no caen en este ciclo, pero realizan la siguiente substitución $\{X/f(\infty)\}$.

Si bien esto parece resolver el problema de eficiencia, generaliza el concepto de término, substitución y unificación al caso del infinito, no considerado en la lógica de Primer Orden, introduciéndo a su vez inconsistencia. Por ejemplo, consideren el siguiente programa definitivo:

```
1 menor(X,s(X)).
2 foo :- menor(S(Y)), Y).
```

El programa expresa que todo número es menor que su sucesor, pero ¿Qué sucede si evaluan foo? Pues que Prolog responderá *Yes*!

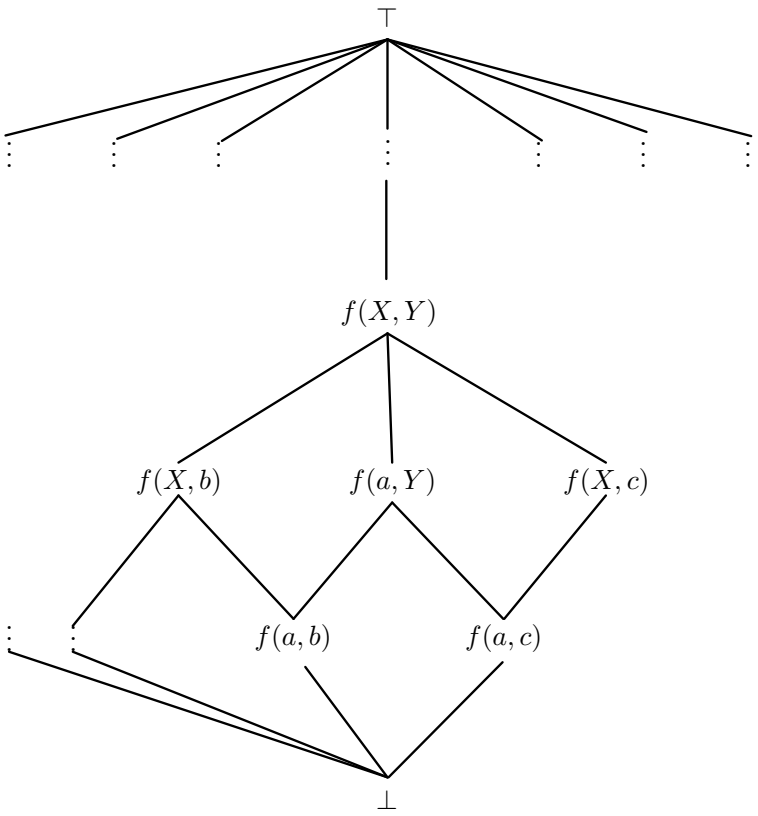


Figura 2.4: Latice de Generalización

La unificación también puede verse como una búsqueda en un espacio conocido como **rejilla de generalización** (Ver figura 2.4), donde se introducen dos términos especiales llamados *top* ($\top$) y *bottom* ($\bot$). En este contexto, la unificación consiste en encontrar la junta más inferior de dos términos en el gráfico [66, 67].

Rejilla de generalización

Ejemplo 2.17. *Unificar los términos $f(X, b)$ y $f(a, Y)$ es posible porque ambos nodos convergen en $f(a, b)$.*

2.3.5 Resolución-SLD

El método de razonamiento descrito informalmente al inicio de esta sesión, puede resumirse con la siguiente regla de inferencia:

$$\frac{\forall\neg(\alpha_1 \wedge \cdots \wedge \alpha_{i-1} \wedge \alpha_i \wedge \alpha_{i+1} \wedge \cdots \wedge \alpha_m) \quad \forall(\beta_0 \leftarrow \beta_1 \wedge \cdots \wedge \beta_n)}{\forall\neg(\alpha_1 \wedge \cdots \wedge \alpha_{i-1} \wedge \beta_1 \wedge \cdots \wedge \beta_n \wedge \alpha_{i+1} \wedge \cdots \wedge \alpha_m)\theta}$$

o, de manera equivalente, usando la notación de los programas definitivos:

$$\frac{\leftarrow \alpha_1, \ldots, \alpha_{i-1}, \alpha_i, \alpha_{i+1}, \ldots, \alpha_m \quad \beta_0 \leftarrow \beta_1, \ldots, \beta_n}{\leftarrow (\alpha_1, \ldots, \alpha_{i-1}, \beta_1, \ldots, \beta_n, \ldots, \alpha_m)\theta}$$

donde:

1. $\alpha_1, \ldots, \alpha_m$ son fbf atómicas.
2. $\beta_0 \leftarrow \beta_1, \ldots, \beta_n$ es una cláusula definitiva en el programa Δ ($n \geq 0$).

3. $MGU(\alpha_i, \beta_0) = \theta$.

La regla tiene dos premisas: una meta y una cláusula definitivas. Observen que cada una de ellas está cuantificada universalmente, por lo que el alcance de los cuantificadores es disjunto. Por otra parte, solo hay un cuantificador universal para la conclusión, por lo que se requiere que el conjunto de variables en las premisas sea disjunto. Puesto que todas las variables en las premisas están cuantificadas, es siempre posible renombrar las variables de la cláusula definitiva para cumplir con esta condición.

La meta definida puede incluir muchas fbf atómicas que unifican con la cabeza de alguna cláusula en el programa. En este caso, es deseable contar con un mecanismo determinista para seleccionar un átomo α_i a unificar. Se asume una función que selecciona una submeta de la meta definida (**función de selección**).

La regla de inferencia presentada es la única necesaria para procesar programas definitivos. Esta regla es una versión de la regla de inferencia conocida como **principio de resolución**, introducido por J.A. Robinson en 1965. El principio de resolución aplica a cláusulas. Puesto que las cláusulas definitivas son más restringidas que las cláusulas, la forma de resolución presentada se conoce como resolución-SLD (resolución lineal para cláusulas definitivas con función de selección).

El punto de partida de la aplicación de esta regla de inferencia es una meta definida G_0:

$$\leftarrow \alpha_1, \ldots, \alpha_m \quad (m \geq 0)$$

De esta meta, una submeta α_i será seleccionada, de preferencia por una función de selección. Una nueva meta G_1 se construye al seleccionar una cláusula del programa $\beta_0 \leftarrow \beta_1, \ldots, \beta_n$ $(n \geq 0)$ cuya cabeza β_0 unifica con α_i, resultando en θ_1. G_1 tiene la forma:

$$\leftarrow (\alpha_1, \ldots, \alpha_{i-1}, \beta_1, \ldots, \beta_n, \ldots, \alpha_m)\theta_1$$

Ahora es posible aplicar el principio de resolución a G_1 para obtener G_2, y así sucesivamente. El proceso puede terminar o no. Hay dos situaciones donde no es posible obtener G_{i+1} a partir de G_i:

1. cuando la submeta seleccionada no puede ser resuelta (no es unificable con la cabeza de una cláusula del programa).
2. cuando $G_i = \square$ (meta vacía = **f**).

Definición 2.28 (Derivación-SLD)**.** *Sea G_0 una meta definitiva, Δ un programa definitivo y $\mathcal{R}$ una función de selección. Una derivación SLD de G_0 (usando Δ y $\mathcal{R}$) es una secuencia finita o infinita de metas:*

$$G_0 \overset{\alpha_0}{\rightsquigarrow} G_1 \ldots G_{n-1} \overset{\alpha_{n-1}}{\rightsquigarrow} G_n$$

Para manejar de manera consistente el renombrado de variables, las variables en una cláusula α_i serán renombradas poniéndoles subíndice i.

Cada derivación SLD nos lleva a una secuencias de MGUs $\theta_1, \ldots, \theta_n$. La composición

$$\theta = \begin{cases} \theta_1\theta_2 \ldots \theta_n & \text{si } n > 0 \\ \epsilon & \text{si } n = 0 \end{cases}$$

de MGUs se conoce como la **substitución computada** de la derivación.

Substitución computada

Ejemplo 2.18. *Consideren la meta definida* $\leftarrow orgulloso(Z)$ *y el programa discutido en la clase anterior.*

$G_0 = \leftarrow orgulloso(Z).$

$\alpha_0 = orgulloso(X_0) \leftarrow padre(X_0, Y_0), recien\_nacido(Y_0).$

La unificación de $orgulloso(Z)$ *y* $orgulloso(X_0)$ *nos da el MGU* $\theta_1 = \{X_0/Z\}$. *Asumamos que nuestra función de selección es tomar la submeta más a la izquierda. El primer paso de la derivación nos conduce a:*

$G_1 = \leftarrow padre(Z, Y_0), recien\_nacido(Y_0).$

$\alpha_1 = padre(X_1, Y_1) \leftarrow papa(X_1, Y_1).$

En el segundo paso de la resolución el MGU $\theta_2 = \{X_1/Z, Y_1/Y_0\}$ *es obtenido. La derivación continua como sigue:*

$G_2 = \leftarrow papa(Z, Y_0), recien\_nacido(Y_0).$

$\alpha_2 = papa(juan, ana).$

$G_3 = \leftarrow recien\_nacido(ana).$

$\alpha_3 = recien\_nacido(ana).$

$G_4 = \square$

la substitución computada para esta derivación es:

$$\theta_1\theta_2\theta_3\theta_4 = \{X_0/Z\}\{X_1/Z, Y_1/Y_0\}\{Z/juan, Y_0/ana\}\epsilon$$

$$= \{X_0/juan, X_1/juan, Y_1/ana, Z/juan, Y_0/ana\}$$

Las derivaciones SLD que terminan en la meta vacía ($\square$) son de especial importancia pues corresponden a refutaciones a la meta inicial (y proveen las respuestas a la meta).

Definición 2.29 (Refutación SLD)**.** *Una derivación SLD finita:*

$$G_0 \overset{\alpha_0}{\rightsquigarrow} G_1 \ldots G_{n-1} \overset{\alpha_{n-1}}{\rightsquigarrow} G_n$$

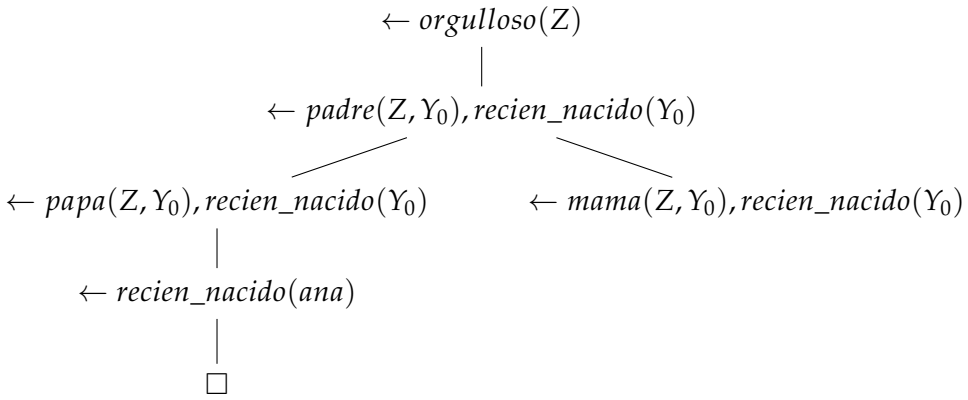
donde $G_n = \square$*, se llama refutación SLD de* G_0.

Definición 2.30 (Derivación fallida)**.** *Una derivación de la meta definitiva* G_0 *cuyo último elemento no es la meta vacía y no puede resolverse con ninguna cláusula del programa, es llamada derivación fallida.*

Definición 2.31 (Arbol-SLD). *Sea Δ un programa definitivo, G_0 una meta definitiva, y $\mathcal{R}$ una función de selección. El árbol-SLD de G_0 (usando Δ y $\mathcal{R}$) es un árbol etiquetado, posiblemente infinito, que cumple las siguientes condiciones:*

- *La raíz del árbol está etiquetada por G_0.*
- *Si el árbol contiene un nodo etiquetado como G_i y existe una cláusula renombrada $\alpha_i \in \Delta$ tal que G_{i+1} es dervidada de G_i y α_i via $\mathcal{R}$, entonces el nodo etiquetado como G_i tiene un hijo etiquetado G_{i+1} El arco que conecta ambos nodos está etiquetado como α_i.*

Por ejemplo:



2.3.6 Propiedades de la resolución-SLD

Definición 2.32 (Robustez). *Sea Δ un programa definitivo, $\mathcal{R}$ una función de selección, y θ una substitución de respuesta computada a partir de Δ y $\mathcal{R}$ para una meta $\leftarrow \alpha_1, \ldots, \alpha_m$. Entonces $\forall((\alpha_1 \wedge \cdots \wedge \alpha_m)\theta)$ es una consecuencia lógica del programa Δ.*

Definición 2.33 (Completez). *Sea Δ un programa definitivo, $\mathcal{R}$ una función de selección y $\leftarrow \alpha_1, \ldots, \alpha_m$ una meta definitiva. Si $\Delta \models \forall((\alpha_1 \wedge \cdots \wedge \alpha_m)\sigma)$, entonces existe una refutación de $\leftarrow \alpha_1, \ldots, \alpha_m$ vía $\mathcal{R}$ con una substitución de respuesta computada θ, tal que $(\alpha_1 \wedge \cdots \wedge \alpha_m)\sigma$ es un caso de $(\alpha_1 \wedge \cdots \wedge \alpha_m)\theta$.*

2.4 LECTURAS Y EJERCICIOS SUGERIDOS

El material aquí presentado está basado principalmente en los textos de Genesereth y Nilsson [30], capítulo 2; y el de Nilsson y Maluszynski [62], capítulo 1. Una lectura complementaria a estos textos son los capítulos 8 y 9 del texto de Russell y Norvig [77]. Una aproximación más computacional a la Lógica de Primer Orden, puede encontrarse en Huth y Ryan [39], capítulo 2.

El algoritmo de unificación presentado (Ver página 59) es una versión del algoritmo propuesto por Robinson [74] en su trabajo sobre el principio de resolución. Knight [45] ofrece una perspectiva multi-disciplinaria al problema de la unificación. Observen que este proceso es relevante en diferentes áreas de las Ciencias de la Computación, como la demostración de teoremas,

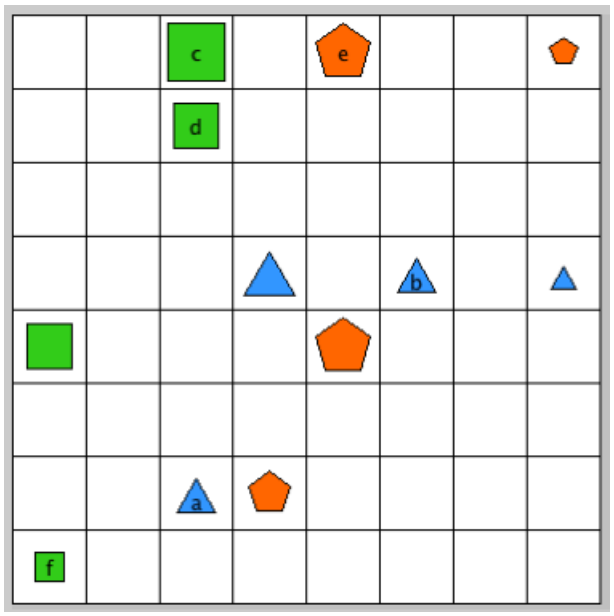


Figura 2.5: Mundo de Tarski para los ejercicios 1 y 2.

la programación lógica, el procesamiento de lenguaje natural, la complejidad computacional y la teoría de computabilidad; de forma que un marco de referencia general sobre este tema es de agradecerse. La resolución lineal con función de selección (resolución-SL) fue introducida por Kowalski y Kuehner [48]. Su restricción a los programas definitivos (resolución-SLD) fue propuesta más tarde por el mismo Kowalski [47].

Ejercicios

Ejercicio 2.1. *Traduzca a Lógica de Primer Orden el enunciado: Todo hijo de mi padre es mi hermano.*

Ejercicio 2.2. *Modifique la definición de la sintaxis de la lógica de primer orden (Def. 2.2), para que incluya variables proposicionales.*

Ejercicio 2.3. *Consideren el Mundo de Tarski [3] mostrado en la figura 2.5. Escriban una interpretación para la escena mostrada, considerando los siguientes predicados con su significado evidente:* triángulo/1, cuadrado/1, pentágono/1, mediano/1, grande/1, pequeño/1, másPequeñoQue/2, aLaIzquierdaDe/2, enLaMismaColumna/2.

Ejercicio 2.4. *En la interpretación del ejercicio anterior, cuales de las siguientes fórmulas son safisfacibles y cuales no:*

1. $triangulo(a) \wedge cuadrado(c) \wedge pentagono(e)$
2. $mediano(a) \wedge \neg grande(a) \wedge \neg pequeno(a) \wedge pequeno(f) \wedge grande(c)$
3. $masPequeno(f,a) \wedge masPequeno(a,c) \wedge aLaIzquierdaDe(f,a) \wedge aLaIzquierdaDe(e,b)$
4. $\forall X \forall Y(aLaIzquierdaDe(X,Y) \vee mismaColumna(X,Y) \vee aLaIzquierdaDe(Y,X))$

5. $\forall X \forall Y(cuadrado(X) \wedge pentagono(Y) \rightarrow aLaIzquierdaDe(X, Y))$
6. $\exists X \exists Y(triangulo(X) \wedge cuadrado(Y) \wedge mismaColumna(X, Y))$

Ejercicio 2.5. *En la misma interpretación, introduzca un predicado que exprese una relación entre tres objetos del universo de discurso. Utilice el predicado introducido en una fórmula bien formada satisfacible.*