4 PRINCIPIO DE RESOLUCIÓN

Este capítulo introduce el mecanismo de inferencia utilizado por la mayoría de los sistemas de programación lógica. Si seguimos considerando Prolog desde la perspectiva de los sistemas formales, hemos descrito ya su lenguaje y su teoría de modelo; ahora describiremos su teoría de prueba. El mecanismo en cuestión es un caso particular de la regla de inferencia llamada **principio de resolución** [21]. La idea es acotar el uso de este principio a programas definitivos, dando lugar a la **resolución-SLD** [13]. Este principio constituye el fundamento de la semántica operacional de los programas definitivos. La resolución-SLD se demostrará correcta con respecto a la teoría del modelo descrita en la clase anterior.

4.1 INTRODUCCIÓN

La programación lógica concierne el uso de la lógica (restringida a cláusulas) para representar y resolver problemas. Este uso es ampliamente aceptado en Inteligencia Artificial (IA), donde la idea se resume como sigue: Un problema o sujeto de investigación puede describirse mediante un conjunto de fórmulas bien formadas (fbf), de preferencia en forma 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 fbf que describen el problema. Por lo tanto, encontrar que fbf ϕ son consecuencia lógica de un conjunto de fbf Δ, es crucial para muchas áreas de la IA, incluyendo la programación lógica. De forma que nos gustaría tener un procedimiento, algorítmico, que nos permita establecer si $\Delta \models \phi$ es el caso, o no. Este es el tema del presente capítulo: un método decidible conocido como principio de resolución [21].

En el caso de la lógica proposicional, la implicación lógica es **decidible**, es decir, existe un algoritmo que puede resolver el problema (contestar si ó no para cada caso particular $\Delta \models \phi$). Si n es el número de átomos distintos que ocurren en estas fbf, el número de interpretaciones posibles es finito, de hecho es 2^n. Un algoritmo para computar $\Delta \models \phi$ simplemente busca si ϕ es verdadero en todos los modelos de Δ. ¿Qué sucede en el contexto de la lógica de primer orden?

La intuición nos dice que el procedimiento de decisión de la lógica proposicional no es adecuado en primer orden, pues en este caso podemos tener una cantidad infinita de dominios e interpretaciones diferentes. Lo que es peor, el teorema de Church [5, 26], muestra que la lógica de primer orden es indecidible:

Teorema 4 (Church) *El problema de si $\Delta \models \phi$, cuando Δ es un conjunto finito arbitrario de fbf, y ϕ es una fbf arbitraria, es **indecidible**.*

Observen que el problema es indecidible para conjuntos arbitrarios de fbf y para una fbf ϕ arbitraria. No existe un algoritmo que en un número finito de pasos, de la respuesta correcta a la pregunta ¿Es ϕ una consecuencia lógica de Δ?

Existen, sin embargo, procedimientos conocidos como **procedimientos de prueba** que pueden ser de gran ayuda para computar este problema. La idea es que cuando es el caso que $\Delta \models \phi$, existen procedimientos que pueden verificarlo en un número finito de pasos. Por ello suele decirse que la lógica de primer orden es semi-decidible. Aunque parecería trivial, siendo que $\Delta \models \phi$, preguntar ¿$\Delta \models \phi$?, en realidad tal trivialidad es aparente. Podemos hacer la pregunta al procedimiento sin que nosotros sepamos que ese es el caso, y obtendremos una respuesta en un número finito de pasos. 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. Esto es infortunado y, peor aún, inevitable.

Esta sesión introduce el procedimiento de prueba utilizado ampliamente en la programación lógica: el principio de resolución propuesto por Robinson [21]. Si bien este procedimiento está orientado a un lenguaje más expresivo, nosotros nos concentraremos en una versión del principio que aplica a programas definidos y se conoce como resolución-SLD [13] (resolución lineal con función de selección para cláusulas definitivas).

4.2 ¿QUÉ ES UN PROCEDIMIENTO DE PRUEBA?

Hasta este momento, hemos abordado informalmente el concepto de procedimiento de prueba como la manera de generar la prueba de que una fbf ϕ es consecuencia lógica de un conjunto de fbf Δ. Las fbf en Δ se conocen como **premisas** y ϕ es la **conclusión de la prueba**.

La prueba suele consistir de un pequeño número de transformaciones en los cuales nuevas fbf son derivadas de las premisas y de fbf previamente derivadas. Derivar una fbf implica construirla a partir de las premisas y otras fbf derivadas, siguiendo alguna regla de inferencia. Toda **regla de inferencia** formaliza alguna forma natural de razonamiento. Por ejemplo, el *modus ponens* es usado comúnmente en matemáticas, su expresión es:

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

donde la línea superior expresa las premisas y la línea inferior la conclusión.

Es posible ligar varias aplicaciones del *modus ponens* para construir una prueba. Por ejemplo, si tenemos el programa lógico $\Delta = \{p(a), q(b) \leftarrow p(a), r(b) \leftarrow q(b)\}$ es posible derivar la fbf $r(b)$ 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 $r(b) \leftarrow q(b)$.

La secuencia anterior es una prueba de que $r(b)$ puede ser derivada de Δ.

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 consistencia (*soundness*). En general un procedimiento de prueba es **consistente** si todas las fbf ψ que pueden ser derivadas de algún conjunto de fbfs Δ usando el procedimiento, son consecuencias lógicas de Δ. En otras palabras, un procedimiento

de prueba es consistente 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 $\Delta \models p(a)$.

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

4.3 PRUEBAS Y PROGRAMAS LÓGICOS

Recordemos que los enunciados en los programas lógicos tienen la estructura general de la implicación lógica:

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

donde $\alpha_0, \ldots, \alpha_n$ son fbfs atómicas y α_0 puede estar ausente (para representar cláusulas meta). Consideren el siguiente programa definitivo Δ que describe un mundo donde los padres de un recién nacido están orgullosos, Juan es el padre de Marta y Marta es una recién nacida:

$$\begin{aligned}
\mathtt{orgulloso}(X) &\leftarrow \mathtt{padre}(X,Y), \mathtt{recien_nacido}(Y).\\
\mathtt{padre}(X,Y) &\leftarrow \mathtt{papa}(X,Y).\\
\mathtt{padre}(X,Y) &\leftarrow \mathtt{mama}(X,Y).\\
\mathtt{papa}(\mathtt{juan},\mathtt{marta}).&\\
\mathtt{recien_nacido}(\mathtt{marta}).&
\end{aligned}$$

Observen que el programa describe únicamente conocimiento positivo, es decir, no especifica quién no está orgulloso. Tampoco que significa para alguien no ser padre.

Supongamos que deseamos contestar la pregunta ¿Quién está orgulloso? Esta pregunta concierne al mundo descrito por nuestro programa, esto es, concierne al modelo previsto para Δ. La respuesta que esperamos es, por supuesto, $\mathtt{juan}$. 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):

$$\leftarrow \mathtt{orgulloso}(Z).$$

que es una abreviatura de $\forall Z \neg \mathtt{orgulloso}(Z)$ (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 satisfacerle, 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 $marta$:

$$papa(juan, marta).$$

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

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

pero el programa contiene el hecho:

$$recien\_nacido(marta).$$

equivalente a $\neg recien\_nacido(marta) \Rightarrow \mathbf{f}$ 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 $p(s_1, \ldots, s_n)$ y una cláusula de la forma $p(t_1, \ldots, t_n) \leftarrow A_1, \ldots A_n$ para encontrar una instancia común de $p(s_1, \ldots, s_n)$ y $p(t_1, \ldots, t_n)$, es decir, una substitución θ que hace que $p(s_1, \ldots, s_n)\theta$ y $p(t_1, \ldots, t_n)\theta$ sean idénticos. Tal substitución se conoce como **unificador**. La nueva meta se construye remplazando el átomo seleccionado en la meta original, por los átomos de la cláusula seleccionada, aplicando θ a todos los átomos obtenidos de esta manera.

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:

$$\leftarrow (A_1, \ldots A_m)\theta_1 \ldots \theta_k$$

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

$$\Delta \models (A_1 \wedge \cdots \wedge A_m)\theta_1 \ldots \theta_k$$

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.

4.4 SUBSTITUCIÓN

Una substitución remplaza variables por términos, 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". Mientras que la cláusula original dice que $V(p(X)) = \mathbf{t}$ y que $V(q(X)) = \mathbf{t}$ para cualquier X en el dominio, la segunda cláusula dice que esto sólo es cierto cuando cuando $V(X) = f(a)$. 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 26 (Substitución) *Una substitución* θ *es un conjunto finito de la forma:*

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

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*. La substitución* θ *se dice se dice de base (*grounded*) si cada término* t_i *es un término base (no incluye variables)..*

La substitución dada por 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 substitucion $\{X/t \in \theta \mid X \in Var\}$.

Ejemplo 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 27 (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 28 *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* $1 \leqslant i \leqslant n$*. 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 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 29 (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 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 30 (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 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 3 *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. $\theta\sigma)\gamma = \theta(\sigma\gamma)$

4.5 UNIFICACIÓN

Uno de los pasos principales en el ejemplo de la sección 4.3, consistió en hacer que dos fbf atómicas se vuelvan sintácticamente equivalentes. Este proceso se conoce como **unificación** y posee una solución algorítmica.

Definición 31 (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 12

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

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

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

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

Definición 32 (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 33 (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.*

Definición 34 (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 35 (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 $\mathtt{mgu}(\alpha, \beta) = \mathtt{fallo}$. Sin embargo, si una forma resuelta $\{X_1 = t_1, \ldots, X_n = t_n\}$ existe, entonces $\mathtt{mgu}(\alpha, \beta) = \{X_1/t_1, \ldots, X_n/t_n\}$. Un algoritmo para encontrar la forma resuelta de un conjunto de ecuaciones es como sigue:

Ejemplo 13 *El conjunto $\{f(X, g(Y)) = f(g(Z), Z)\}$ tiene una forma resuelta, puesto que:*

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

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

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

Algoritmo 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 14 *El conjunto* $\{f(X, g(X), b) = f(a, g(Z), Z)\}$ *no tiene forma resuelta, puesto que:*

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

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

$$\Rightarrow \{X = a, a = Z, b = Z\}$$

$$\Rightarrow \{X = a, Z = a, b = Z\}$$

$$\Rightarrow \{X = a, Z = a, b = a\}$$

$$\Rightarrow \mathtt{fallo}$$

Ejemplo 15 *El conjunto* $\{f(X, g(X)) = f(Z, Z)\}$ *no tiene forma resuelta, puesto que:*

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

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

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

$$\Rightarrow \mathtt{fallo}$$

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 $\mathtt{subtermino}(X, t)$ (**verificación de ocurrencia**) 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.

4.6 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 ($n \geqslant 0$).
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:

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

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 $\beta_0 \leftarrow \beta_1, \ldots, \beta_n$ ($n \geqslant 0$) 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 36 (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.

Ejemplo 16 *Consideren la meta definida $\leftarrow$ $\mathtt{orgulloso}(Z)$ y el programa discutido en la clase anterior.*

$G_0 = \leftarrow \mathtt{orgulloso}(Z)$.

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

La unificación de $\mathtt{orgulloso}(Z)$ *y* $\mathtt{orgulloso}(X_0)$ *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:*

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

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

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:*

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

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

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

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

$G_4 = \square$

la substitución computada para esta derivación es:

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

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

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 37 (Refutación SLD) *Una derivación SLD finita:*

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

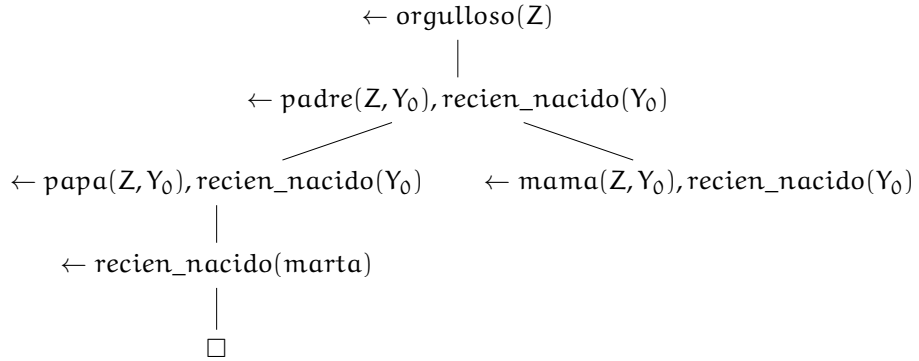
donde $G_n = \square$*, se llama refutación SLD de* G_0*.*

Definición 38 (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 39 (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:



4.7 PROPIEDADES DE LA RESOLUCIÓN-SLD

Definición 40 (Consistencia) *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 41 (Compleción) *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 $(\alpha_1 \wedge \cdots \wedge \alpha_m)\theta$.*