Notas de Clase para IL

5. Deducción en Lógica de Primer Orden

Rafel Farré, Robert Nieuwenhuis,
Pilar Nivela, Albert Oliveras, Enric Rodríguez, Josefina Sierra

12 de febrero de 2008

1. Formas normales y cláusulas

En lógica de primer orden, las definiciones de literal, CNF, DNF, cláusula, cláusula vacía, cláusula de Horn, etc., son iguales a las de lógica proposicional (véanse las *Notas de Clase para IL 3.*), excepto que, en vez de símbolos de predicado de aridad cero, ahora hay átomos cualesquiera.

Así, recordemos: una *cláusula* es una disyunción de literales, es decir, una fórmula de la forma $l_1 \vee \ldots \vee l_m$ con $m \geq 0$ donde cada l_i es un literal, o, equivalentemente, una disyunción $A_1 \vee \ldots \vee A_p \vee \neg B_1 \vee \ldots \vee \neg B_n$ de p literales positivos y n negativos, con $p + n \geq 0$, donde las A_i y B_j son átomos.

Todas las variables de una cláusula están universalmente cuantificadas. Así, si C es una cláusula cuyas variables son $x_1, \ldots, x_k$, es como si tuviéramos la cláusula $\forall x_1 \ \ldots \ \forall x_k \ C$. Pero normalmente estos cuantificadores no se escriben.

Para lo que viene a continuación, necesitamos definir qué es una *subfórmula* de una fórmula F:

- toda fórmula es subfórmula de sí misma
- todas las subfórmulas de F lo son de $\forall x\, F$, $\exists x\, F$, y de $\neg F$
- todas las subfórmulas de F y de G lo son de $(F \vee G)$ y de $(F \wedge G)$.

Si en una fórmula F tenemos una subfórmula $\forall x\, F_1$, que a su vez tiene a $\exists y\, G$ como subfórmula, decimos que *y se encuentra en el ámbito de x en F*, o que *y depende de x*.

1.1. Transformación a forma clausal

Toda fórmula de lógica de primer orden F puede ser transformada en un conjunto (conjunción) de cláusulas S *preservando la satisfactibilidad:* F es satisfactible si y sólo si lo es S. Esta transformación consta de los siguientes pasos, donde el único paso que no preserva la equivalencia lógica es el de la Skolemización:

1. **Movimiento de las negaciones hacia dentro**, aplicando las reglas:

   $$\begin{array}{lcl} \neg(F \wedge G) & \Rightarrow & \neg F \vee \neg G \\ \neg(F \vee G) & \Rightarrow & \neg F \wedge \neg G \\ \neg\neg F & \Rightarrow & F \\ \neg\exists x F & \Rightarrow & \forall x \neg F \\ \neg\forall x F & \Rightarrow & \exists x \neg F \end{array}$$

   Después de este paso aplicando exhaustivamente las reglas (es decir, hasta que no se puedan aplicar más), todas las negaciones estarán aplicadas directamente a los átomos.

2. **Eliminación de conflictos de nombre:**

   En una fórmula (cerrada) F la variable *x tiene un conflicto de nombre* si hay al menos dos subfórmulas de F de la forma $Q_1 x G_1$ y $Q_2 x G_2$ y donde Q_1 y Q_2 son cuantificadores ($\forall$ o $\exists$).

Claramente, toda fórmula F se puede convertir en una fórmula lógicamente equivalente sin conflictos de nombre, introduciendo variables *frescas* (variables que no aparecen en F). Se reemplazan subfórmulas de la forma QxG por $Qx'G'$, donde G' se obtiene sustituyendo en G todas las apariciones de x por la variable fresca x'.

Por ejemplo, $\forall x\, P(x) \wedge \forall x\, \neg Q(x)$ se puede convertir en la fórmula equivalente: $\forall x\, P(x) \wedge \forall y\, \neg Q(y)$.

Es necesario comenzar este reemplazamiento por las subfórmulas más internas. Por ejemplo, en $\forall x\ (\ P(x) \wedge \exists x \neg Q(x)\ )$ hay que comenzar por $\exists x \neg Q(x)$.

Después de estos dos primeros pasos, la fórmula tendrá todas las negaciones aplicadas directamente a los átomos, y no tendrá conflictos de nombre.

3. **[Opcional:] Movimiento de cuantificadores hacia dentro, mientras sea posible**, aplicando las reglas:

$$\begin{array}{lcll} \forall x(F \vee G) & \Rightarrow & \forall xF \ \vee G & \text{Si } x \text{ no aparece en } G \\ \forall x(F \wedge G) & \Rightarrow & \forall xF \ \wedge G & \text{Si } x \text{ no aparece en } G \\ \exists x(F \vee G) & \Rightarrow & \exists xF \ \vee G & \text{Si } x \text{ no aparece en } G \\ \exists x(F \wedge G) & \Rightarrow & \exists xF \ \wedge G & \text{Si } x \text{ no aparece en } G \end{array}$$

Nótese que aplicar estas reglas a fórmulas donde x no aparece en G preserva la equivalencia, y que, por conmutatividad de $\vee$ y de $\wedge$, también tenemos reglas como

$$\exists x(F \wedge G) \Rightarrow F \wedge \ \exists xG \quad \textbf{Si } x \text{ no aparece en } F.$$

Este paso (llamado *miniscoping* porque reduce el ámbito (*scope*) de los cuantificadores) no es imprescindible, pero a menudo ayuda a producir fórmulas más sencillas en el siguiente paso, la Skolemización.

4. **Eliminación de cuantificadores existenciales o *Skolemización***:

Un paso de *Skolemización* de F consiste en reemplazar una subfórmula $\exists y\, G$ por otra G' sin la variable y. Concretamente, G' se obtiene a partir de G reemplazando todas las apariciones de y por un término t, tal que:

- t es una constante fresca c_y, si y no se encuentra en el ámbito de ninguna variable universalmente cuantificada.
- t es $f_y(x_1, \ldots, x_n)$, donde f_y es un símbolo de función fresco, si $\{x_1, \ldots, x_n\}$ es el conjunto no-vacío de las variables universalmente cuantificadas en cuyo ámbito se encuentra y.

Ejemplo 1: Intuitivamente, la idea es que una fórmula $\forall x \exists y\ p(x, y)$ es satisfactible si y sólo si, $\forall x\ p(x, f_y(x))$ es satisfactible, es decir, si existe una función que interprete el símbolo f_y de manera que "escoge" la y adecuada para cada x.

En cambio, una fórmula $\exists y \forall x\ p(x, y)$ es satisfactible si y sólo si, $\forall x\ p(x, c_y)$ es satisfactible, es decir, si en el dominio existe un único elemento c_{y_I} (la interpretación en I de c_y) que sirve para todas las x.

Ejemplo 2: Podemos decir que todo ser humano tiene madre:

$\forall y(\neg humano(y) \vee \exists x\ esmadre(y, x))$.

Esta fórmula se tranformaría en:

$\forall y(\neg humano(y) \vee esmadre(y, f_x(y)))$

Aquí, intuitivamente, el símbolo de función f_x denota la función "madre-de".

Después de los tres primeros pasos y de aplicar exhaustivamente (es decir, mientras sea posible) la Skolemización, la fórmula tendrá todas las negaciones aplicadas directamente a los átomos, no tendrá conflictos de nombre, y sólo tendrá cuantificadores universales.

5. **Movimiento de cuantificadores universales hacia fuera**, aplicando las reglas:

$$(\forall xF) \vee G \quad \Rightarrow \quad \forall x(F \vee G)$$
$$(\forall xF) \wedge G \quad \Rightarrow \quad \forall x(F \wedge G)$$

Después de aplicar exhaustivamente estas reglas (que son correctas puesto que en fórmulas sin conflicto de nombre x no aparecerá en G), la fórmula tendrá todas las negaciones aplicadas directamente a los átomos, no tendrá conflictos de nombre, y sólo tendrá cuantificadores universales, que estarán al principio.

6. **Distribución de $\wedge$ sobre $\vee$**, aplicando (exhaustivamente) la regla:

$(F \wedge G) \vee H \Rightarrow (F \vee H) \wedge (G \vee H)$

Nótese que, teniendo en cuenta la conmutatividad de $\vee$, en realidad también tenemos la regla $F \vee (G \wedge H) \Rightarrow (F \vee G) \wedge (F \vee H)$.

Después de este último paso, tenemos una expresión de la forma:

$$\forall x_1 \ldots \forall x_k\ (l_{11} \vee \ldots \vee l_{1n_1})\ \wedge \ldots \wedge\ (l_{m1} \vee \ldots \vee l_{mn_m})$$

donde cada l_{ij} es un literal. Esta expresión puede ser vista como el conjunto (la conjunción) de m cláusulas:

$$\begin{array}{ll} \forall x_1 \ldots \forall x_k & l_{11} \vee \ldots \vee l_{1n_1} \\ \quad \ldots & \quad \ldots \\ \forall x_1 \ldots \forall x_k & l_{m1} \vee \ldots \vee l_{mn_m} \end{array}$$

donde, puesto que $\forall xF \wedge \forall xG \equiv \forall x(F \wedge G)$, equivalentemente, hemos colocado en cada cláusula los cuantificadores. Pero debido a que se sabe y se asume que en una cláusula todas las variables están universalmente cuantificadas, normalmente los cuantificadores no se escriben. Al igual que en la lógica proposicional, hablaremos de *conjuntos* de cláusulas (y de literales) por la asociatividad, conmutatividad e idempotencia de $\wedge$ (y $\vee$).

1.2. Unificación

Esta subsección trata de la *unificación* de términos. Consideramos sólo términos porque no es relevante la distinción entre si lo que se unifica son términos o átomos.

Sustituciones: Una *sustitución* es un conjunto de pares $\{x_1 = t_1, \ldots, x_n = t_n\}$, donde las x_i son variables distintas y donde las t_i son términos. El conjunto de variables $\{x_1, \ldots, x_n\}$ es el *dominio* de esta sustitución.

A partir de ahora denotaremos los términos por s o por t, las variables por x, las sustituciones por σ (la letra griega sigma), todos ellos posiblemente con subíndices, y el dominio de una sustitución σ se denotará por $Dom(\sigma)$.

Aplicación y composición de sustituciones: Dada una sustitución σ de la forma $\{x_1 = t_1, \ldots, x_n = t_n\}$, y un término t, el término $t\sigma$ es el que se obtiene al reemplazar simultáneamente cada variable x_i en t por el término t_i correspondiente. Similarmente, definimos la aplicación de sustituciones a átomos o a cláusulas.

Si σ y σ' son dos sustituciones $\{x_1 = s_1, \ldots, x_n = s_n\}$ y $\{y_1 = t_1, \ldots, y_m = t_m\}$, la *composición* de σ y σ' es una nueva sustitución $\sigma\sigma'$ que es:

$\{x_1 = s_1\sigma', \ldots, x_n = s_n\sigma'\} \cup \{\, y_i = t_i \mid i \in 1 \ldots m,\ y_i \notin Dom(\sigma) \,\}$.

Nótese que para todo término t, tenemos $t\sigma\sigma' = (t\sigma)\sigma'$, es decir, aplicar la composición es lo mismo que aplicar primero σ y después σ'.

Unificación y unificadores: Dos términos s y t son *unificables* si existe una sustitución σ tal que $s\sigma$ y $t\sigma$ son el mismo término. En este caso σ es un *unificador* de s y t.

El unificador más general: El *unificador más general* σ de s y t, escrito $\sigma = mgu(s, t)$, es un unificador de s y t tal todo unificador σ' de s y t es un caso particular suyo, es decir, existe una sustitución σ'' tal que $\sigma' = \sigma\sigma''$.

Nota: $mgu(s, t)$ es único salvo cambios de nombre de variables equivalentes. Por ejemplo, podemos expresar el *mgu* de $g(f(x), x)$ y $g(y, z)$ como $\{y = f(x), z = x\}$, o también como $\{y = f(z), x = z\}$.

Unificador simultáneo: En vez de trabajar sobre un solo problema de unificación $s = t$ en el que se busca $mgu(s, t)$, partiremos de un problema más general: un conjunto de pares de términos $\{s_1 = t_1, \ldots, s_n = t_n\}$ para el que se busca un *mgu σ simultáneo*, es decir, tal que $s_i\sigma$ es $t_i\sigma$ para todo $i \in 1 \ldots n$. Nótese que en un problema de unificación una expresión $s = t$ es equivalente a $t = s$; el orden entre los dos términos no importa.

Variables resueltas y problemas de unificación resueltos: En un problema de unificación de la forma $P \cup \{x = t\}$, decimos que x es una variable *resuelta* si no aparece en P ni en t, es decir, aparece una sola vez en todo el problema.

Un problema de unificación P de la forma $\{x_1 = t_1, \ldots, x_n = t_n\}$, donde todas las x_i son variables resueltas, se dice que está *resuelto*.

No resulta difícil de ver que encontrar el *mgu* de un problema resuelto P es trivial: el *mgu* es el propio P.

Algoritmo de unificación basado en reglas: Considera el siguiente conjunto de reglas de transformación que, dado un problema inicial $P_0 = \{s_1 = t_1, \ldots, s_n = t_n\}$, o bien lo convierte en un problema resuelto o bien falla indicando que no hay unificador

simultáneo:

$$
\begin{array}{ll}
P \cup \{t = t\} & \Rightarrow P \\
P \cup \{f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)\} & \Rightarrow P \cup \{t_1 = s_1, \ldots, t_n = s_n\} \\
P \cup \{f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)\} & \Rightarrow \text{fallo} \\
 & \quad \textbf{Si } f \neq g \\
P \cup \{x = t\} & \Rightarrow P\{x = t\} \cup \{x = t\} \\
 & \quad \textbf{Si } x \in vars(P) \text{ y } x \notin vars(t) \\
 & \quad \text{y } t \text{ no es variable resuelta en} \\
 & \quad P \cup \{x = t\} \\
P \cup \{x = t\} & \Rightarrow \text{fallo} \\
 & \quad \textbf{Si } x \in vars(t) \text{ y } x \neq t
\end{array}
$$

Nótese que la segunda y la tercera regla también se aplican al caso donde f y/o g son constantes (es decir, si n y/o m son 0). Por ejemplo, tenemos $P \cup \{a = a\} \Rightarrow P$, y también tenemos $P \cup \{a = c\} \Rightarrow$ fallo si a y c son símbolos de función constantes (es decir, de aridad cero) distintas y también $P \cup \{a = f(x)\} \Rightarrow$ fallo.

Nótese que en la cuarta regla $P\{x = t\}$ es la aplicación a P de la sustitución $\{x = t\}$.

Este algoritmo funciona, es decir, acaba en fallo si no hay unificador, y acaba con el mgu si lo hay. Esto es cierto porque:

1. Cualquier estrategia de aplicación de las reglas termina, es decir, no existe ningún problema finito P_0 con una secuencia infinita de aplicaciones de reglas $P_0 \Rightarrow P_1 \Rightarrow P_2 \Rightarrow \ldots$

2. Un problema al que ya no se le puede aplicar ninguna regla está resuelto.

3. Las reglas transforman cualquier problema de unificación en otro *equivalente*, en el sentido de que tiene el mismo conjunto, posiblemente vacío, de unificadores.

Los apartados 2 y 3 son fáciles de ver. Para demostrar 1, se sabe que las demostraciones de terminación *de cualquier programa, proceso, o máquina* siempre pueden descomponerse en dos pasos:

- la definición de un orden *bien fundado* $\succ$ sobre el conjunto de *estados* del proceso ($\succ$ es bien fundado si no existe ninguna secuencia infinita de la forma $a_0 \succ a_1 \succ a_2 \succ \ldots$).

- la demostración de que, en cada paso del proceso, el estado decrece con respecto a $\succ$.

En nuestro algoritmo, el estado es el problema de unificación simultánea que tenemos en cada momento. Un orden que funciona es un orden lexicográfico sobre pares de naturales: la primera componente es el número de variables no-resueltas y la segunda es el tamaño (número de símbolos) del problema. Todas las reglas reducen la primera componente o bien reducen la segunda sin cambiar la primera.

Mucho más que un solo algoritmo. Este tipo de reglas resultan muy convenientes porque en realidad no se proporciona un solo algoritmo, sino toda una *familia de algoritmos*. Una *estrategia* utilizada en la aplicación de las reglas define en cada momento

qué regla se aplica, y sobre qué pareja $s = t$ del problema, y cada estrategia distinta da lugar a un algoritmo distinto.

Por ejemplo, determinada estrategia, implementada con cuidado, nos dará el conocido algoritmo de Robinson, otra nos dará el de Martelli y Montanari, otras nos darán algoritmos de coste exponencial. Además, los resultados 1-3 nos dan la *corrección de cualquier estrategia* de éstas.

Este tipo de algoritmos es muy importante porque la unificación es la operación fundamental (tan básica como la suma en la aritmética) en la resolución y sus aplicaciones a la demostración automática, la programación lógica o las bases de datos deductivas.

1.3. Resolución y factorización

Haciendo uso de la unificación, podemos definir las reglas de deducción de resolución y factorización. Véanse las *Notas de Clase para IL 3.*, acerca de las nociones de reglas deductivas, y propiedades como la corrección y la completitud. En las reglas siguientes, A y B denotan átomos (literales positivos) y C y D denotan cláusulas (posiblemente vacías). En la resolución, siempre se asume que las dos premisas *no comparten variables;* si no es así, antes de aplicar resolución hay que realizar los renombramientos necesarios:

(i) *resolución:*

$$\frac{A \vee C \qquad \neg B \vee D}{C\sigma \vee D\sigma} \qquad \text{donde } \sigma = mgu(A, B)$$

(ii) *factorización:*

$$\frac{A \vee B \vee C}{A\sigma \vee C\sigma} \qquad \text{donde } \sigma = mgu(A, B)$$

La resolución y la factorización son correctas. La resolución en lógica de primer orden es la extensión de su versión de lógica proposicional para literales con variables.

Intuitivamente, la resolución en lógica de primer orden es correcta, porque si $I \models \forall x_1 \ldots \forall x_n\ (A \vee C)$, en particular tendremos $I \models A\sigma \vee C\sigma$, y similarmente para la otra premisa, $I \models \neg B\sigma \vee D\sigma$. Pero puesto que $A\sigma$ y $B\sigma$ son el mismo átomo, igual que en el caso proposicional tendremos $C\sigma \vee D\sigma$ como consecuencia lógica (esto sólo es una idea intuitiva, no una demostración).

Un razonamiento similar da intuición sobre la corrección de la regla de factorización: si $I \models A \vee B \vee C$, en particular tendremos $I \models A\sigma \vee B\sigma \vee C\sigma$, y como $A\sigma$ y $B\sigma$ son el mismo átomo, por idempotencia del $\vee$ tendremos $I \models A\sigma \vee C\sigma$.

La resolución y la factorización no son completas. Como en el caso proposicional: por ejemplo, tenemos $p \models p \vee q$, pero no podemos obtener $p \vee q$ a partir de p mediante estas reglas deductivas.

La resolución y la factorización sí son refutacionalmente completas: Si S es un conjunto de cláusulas de primer orden sin igualdad que es insatisfactible, entonces la

cláusula vacía $\square$ pertenece a $ResFact(S)$, la clausura de S bajo resolución y factorización.

Para la lógica de primer orden con igualdad, la resolución y la factorización son incompletas refutacionalmente. Es necesaria una regla adicional, llamada *paramodulación*, de la que aquí no hablaremos más.

La resolución y la factorización pueden no terminar: Al contrario que en la lógica proposicional, para un conjunto finito de cláusulas S, aquí la clausura de S bajo resolución y factorización puede ser infinita.

Por ejemplo, si S tiene la cláusula $p(a)$ y la cláusula $\neg p(x) \vee p(f(x))$, entonces $ResFact(S)$ contiene infinitas cláusulas:

$p(a),\ p(f(a)),\ p(f(f(a))),\ p(f(f(f(a)))), \ldots$

Uso de las reglas deductivas: Como en la lógica proposicional, problemas como determinar si una fórmula es tautología, o satisfactible, o la consecuencia o equivalencia lógica entre fórmulas, etc., pueden ser transformados a un problema de satisfactibilidad de un conjunto de cláusulas. Por ejemplo:

$F \models G$	si, y sólo si,
$F \wedge \neg G$ es insatisfactible	si, y sólo si,
S, la forma clausal de $F \wedge \neg G$, es insatisfactible,	si, y sólo si,
$\square \in ResFact(S)$	

donde $ResFact(S)$ es la clausura bajo resolución y factorización de S (como ya hemos dicho, este último "si, y sólo si" sólo se cumple en la LPO sin igualdad).

SAT en lógica de primer orden no es decidible. En las *Notas de Clase para IL 2 y 3* vimos que el problema de SAT para lógica proposicional era *decidible*: existe un programa (un programa de ordenador, escrito en un lenguaje de programación) que toma como entrada una fórmula proposicional arbitraria F, y

- si F **es** satisfactible, siempre acaba con salida "sí".
- si F **no es** satisfactible, siempre acaba con salida "no".

Por ejemplo, un programa para SAT en lógica proposicional puede construir la tabla de verdad y evaluar F en cada interpretación que existe. Otra opción es, puesto que la resolución en lógica proposicional termina, calcular la clausura bajo resolución y ver si está la cláusula vacía o no. Estos métodos pueden ser costosos en tiempo de cómputo, pero, con suficientes recursos de tiempo y memoria, siempre terminan y contestan correctamente. A estos programas se les llama *procedimientos de decisión*.

Al contrario de lo que ocurre en la lógica proposicional, está demostrado que no puede existir ningún procedimiento de decisión para SAT en lógica de primer orden: se dice que *SAT en lógica de primer orden es indecidible*.

SAT en lógica de primer orden es co-semi-decidible: En cambio, sí existe un programa que toma como entrada una fórmula arbitraria F de lógica de primer orden, y:

- si F **es** satisfactible, o bien no acaba, o bien acaba con salida "sí"
- si F **no es** satisfactible, siempre acaba con salida "no"

Por ejemplo, el programa puede pasar F a forma clausal S_0, e ir calculando la clausura bajo resolución y factorización (y, en el caso de la LPOI, bajo paramodulación) de S_0 por niveles: $S_1, S_2, \ldots$ Si F es insatisfactible, la cláusula vacía aparecerá (al cabo de un tiempo finito) en algún S_j y podemos acabar con salida "no". Si la resolución y la factorización acaban sin generar la cláusula vacía, podemos acabar con salida "sí". Si la resolución no acaba, el programa no acaba. Puesto que existe tal programa, se dice que el problema de SAT en lógica de primer orden es *co-semi-decidible*.

Problemas semi-decidibles: También existe un programa que toma como entrada una fórmula arbitraria F de lógica de primer orden, y:

- si F **es** una tautología, siempre acaba con salida "sí"
- si F **no es** una tautología, o bien no acaba, o bien acaba con salida "no"

Un programa así es un *procedimiento de semi-decisión* para el problema de determinar si una fórmula de lógica de primer orden es una tautología. Este programa puede obtenerse así: puesto que el problema es equivalente a la insatisfactibilidad de $\neg F$, puede utilizar resolución como procedimiento de co-semi-decisión para SAT. De la misma manera, el problema de determinar para dos fórmulas dadas F y G, si $F \models G$, también es semi-decidible.

2. Ejercicios

1. (Dificultad 2) Transforma a forma clausal las siguientes fórmulas:

   a) $\forall x\ (\ \forall y\ (p(y) \rightarrow q(x,y)) \rightarrow \exists y\ q(y,x)\ \ )$

   b) $\forall y\ (\ \neg p(y) \rightarrow \forall y \exists x\ q(y,x)\ )$

   c) $\exists x\ \forall y\ (\ \ \forall z\ (p(y,z) \vee x \neq y)\ \ \rightarrow\ \ (\forall z\ q(y,z) \wedge \neg r(x,y))\ \ )$

2. (Dificultad 2) Demuestra que la Skolemización no preserva la equivalencia lógica.
   Ayuda: considera $\forall x \exists y\ p(x,y)$ y su transformación $\forall x\ p(x, f(x))$, y define una I de dos elementos sobre los símbolos f y p.

3. (Dificultad 1) Unifica $p(\ f(x, g(x)),\ h(y),\ v\ )$ con $p(\ y,\ h(v),\ f(g(z), w)\ )$.
   Calcula un unificador más general y otro que no sea el más general (si hay).

4. (Dificultad 1) Sean los términos:

   t_1 : $f(x, h(g(x)), x')$
   t_2 : $f(a, y, y)$
   t_3 : $f(z, h(z), h(b))$

   ¿Cuáles de los pares (t_i, t_j) son unificables? Da un *mgu* si hay.

5. (Dificultad 2) Demuestra por resolución la insatisfactibilidad del conjunto de cláusulas:

   1 : $p(a, z)$
   2 : $\neg p(f(f(a)), a)$
   3 : $\neg p(x, g(y)) \vee p(f(x), y)$

6. (Dificultad 3) Demuestra que es importante que no haya ninguna variable x que aparezca, con ese nombre, en los dos literales que se unifican cuando se hace resolución.
   Ayuda: Da un ejemplo de incompletitud refutacional de la regla de resolución "sin renombramiento previo de variables".

7. (Dificultad 3) (Schöning, Exercise 85) Formaliza los siguientes hechos:

   (a) *"Todo dragón está feliz si todos sus hijos pueden volar"*

   (b) *"Los dragones verdes pueden volar"*

   (c) *"Un dragón es verde si es hijo de al menos un dragón verde"*

   Demuestra por resolución que la conjunción de (a), (b) y (c) implica que:

   (d) *"Todos los dragones verdes son felices"*

8. (Dificultad 2) Demuestra por resolución que es válida la fórmula:

   $\forall x \exists y \ (\ p(f(f(x)), y) \ \wedge \ \forall z \ (\ p(f(x), z) \rightarrow p(x, g(x, z)) \ ) \ ) \ \rightarrow \ \forall x \exists y \ p(x, y)$

9. (Dificultad 3) Demuestra que la resolución sin la factorización no es refutacionalmente completa.
   Ayuda: da un contraejemplo de dos cláusulas con dos literales cada una.

10. (Dificultad 3) Una cláusula es *de base* si no contiene variables. Demuestra que la satisfactibilidad de conjuntos de cláusulas de base sin igualdad es decidible.

11. (Dificultad 2) Considera la regla de *resolución unitaria:*

$$\frac{A \qquad \neg B \vee C}{C\sigma} \qquad \text{donde } \sigma = mgu(A, B)$$

    Esta regla es refutacionalmente completa para cláusulas de Horn sin igualdad. Demuestra que no lo es sin la restricción a cláusulas de Horn.

12. (Dificultad 4) Sea S un conjunto de cláusulas de Horn sin igualdad donde todos los símbolos de función que aparecen son de aridad cero (son símbolos de constante). Observa que puede haber símbolos de predicado de todo tipo.

    a) Demuestra que la resolución unitaria termina para S (es decir, la clausura bajo resolución unitaria es finita).

    b) Utilizando este hecho, y la completitud refutacional mencionada en el ejercicio previo, demuestra que la satisfactibilidad de conjuntos de Horn S sin símbolos de función de aridad mayor que cero es decidible.

    Nota: las cláusulas de este tipo forman el lenguage del *Datalog*, y este resultado de decidibilidad hace que el *Datalog* sirva para bases de datos deductivas.

13. (Dificultad 3) Escribe un conjunto de cláusulas S sin símbolos de función para el que la resolución no termina.