Demostración Automática de Teoremas y la Nutria

José Alfredo Amor Montaño y
Favio Ezequiel Miranda Perea
Departamento de Matemáticas
Facultad de Ciencias UNAM
Circuito Exterior, Ciudad Universitaria
04510, México D.F., México
jaam@hp.fciencias.unam.mx
fmiranda@ada.fciencias.unam.mx

Resumen

Se presentan dos procedimientos generales de demostración automática de teoremas para cualquier teoría finitamente descrita, expresable en un lenguaje de predicados con igualdad. Se describe además una implementación parcial del primer procedimiento y una implementación completa y muy eficiente del segundo procedimiento. Estos procedimientos están basados únicamente en lógica matemática clásica y necesariamente serán correctos y completos para los teoremas de la teoría en cuestión por resultados púramente lógicos. Es decir, si **P** es un enunciado cualquiera del lenguaje de predicados con igualdad de la teoría en cuestión, entonces:

P es un teorema de la teoría si y sólo si el procedimiento así lo afirma.

1 Introducción

Las décadas de los 70 y siguientes han sido marcadas por una revolución en el desarrollo de las matemáticas y otras disciplinas que dependen de ellas; debido a la presencia de computadoras digitales de gran memoria y muy alta velocidad. En parte, se ha hecho realizable la propuesta de Leibniz de un cálculo universal de razonamiento para llevar a cabo pruebas matemáticas en forma mecánica. En lo que sigue trabajaremos con el lenguaje de predicados con igualdad o lenguaje de primer orden con igualdad. Se trata de un lenguaje formal de poderosa expresividad y precisión, en el cual se pueden formalizar la mayoría de las teorías matemáticas; tiene además la cualidad de ser fácil de manipular mecánicamente, tarea que se puede realizar con una máquina computadora. Para ello se cuenta con dos procedimientos. El primero de ellos tiene como base el Teorema de Herbrand y el segundo el método de

[0] *Clasificación AMS:* 03B35,03B70,68T15,68T20

resolución; ambos aplicables a la lógica de primer orden con igualdad. Usaremos las letras griegas $\varphi, \alpha, \beta, \gamma$ ó $\alpha_1, \alpha_2, \ldots, \alpha_n$ para referirnos a fórmulas de un lenguaje de primer orden con igualdad. Por ejemplo, si el predicado P simboliza una relación de dos argumentos, el símbolo f una operación de dos argumentos, y la constante c un elemento distinguido, entonces los enunciados α, β, γ, podrían ser:

$$\forall x \forall y (Pxy \rightarrow Pyx), \forall x \forall y \forall z \left[f\left(f\left(x, y\right), z\right) = f\left(x, f\left(y, z\right)\right)\right], \\ \forall w \left(f\left(w, c\right) = w\right)$$

que expresan, en ese lenguaje que la relación es simétrica, que la operación es asociativa y que el elemento distinguido es neutro para la operación, respectivamente.

Si Σ es un conjunto de enunciados perteneciente a un lenguaje de predicados, y si pensamos en Σ como un conjunto de hipótesis, como axiomas de una teoría o como una base de datos, y si β es un enunciado particular del lenguaje, es natural preguntarnos:

¿ Es β consecuencia lógica de Σ ? o
¿ β se sigue de Σ, lógicamente ? o bien
¿ β se puede deducir (es teorema) a partir de Σ ?

Lo anterior lo denotaremos de la siguiente manera:

$\Sigma \models \beta$ "β es consecuencia lógica de Σ".
$\Sigma \vdash \beta$ "β es deducible (es teorema) a partir de Σ".

En la segunda expresión se sobreentiende que nos referimos a un sistema formal o Cálculo de Predicados (CP), que cumple con el Teorema de Completud de la Lógica (Gödel 1930):

$$\Sigma \vdash \beta \Leftrightarrow \Sigma \models \beta$$

Si Σ es finito, digamos $\Sigma = \{\alpha_1, ..., \alpha_n\}$ entonces en vez de $\Sigma \vdash \beta$ o $\Sigma \models \beta$ escribimos

$$\alpha_1, ..., \alpha_n \vdash \beta \Leftrightarrow \alpha_1, ..., \alpha_n \vDash \beta$$

Y el problema que nos hemos planteado consiste en responder a la pregunta

$$¿\alpha_1, ..., \alpha_n \models \beta \ ?$$

Algunos ejemplos de conjuntos Σ con un número finito de enunciados son: la teoría de órdenes lineales, la teoría de órdenes densos sin extremos, la teoría de grupos, la aritmética recursiva y, en general, cualquier conjunto finito de axiomas o cualquier base de datos finita dada en lenguaje de predicados con igualdad. Es importante precisar que el concepto "β es una consecuencia lógica de $\alpha_1, ..., \alpha_n$", (denotado por $\alpha_1, ..., \alpha_n \models \beta$) significa que para toda interpretación respecto a la cual $\alpha_1, ..., \alpha_n$ sean verdaderos, β sea verdadero también. Llamamos modelo de un conjunto de enunciados

a cualquier interpretación respecto a la cual el conjunto de enunciados es verdadero. Con esta precisión de lo que significa la palabra modelo en lóg ica, podemos definir " β es una consecuencia lógica de $\alpha_1, ..., \alpha_n$" si y sólo si todo modelo de $\alpha_1, ..., \alpha_n$ es un modelo de β. Es decir, para cualquier interpretación, β es verdadera cada vez que $\alpha_1, ..., \alpha_n$ son verdaderas.

Teorema 1.1 *(Teorema Básico)*

$$\alpha_1, ..., \alpha_n \vDash \beta \Leftrightarrow (\alpha_1 \wedge ... \wedge \alpha_n \wedge \neg\beta) \text{ no tiene modelo.}$$

Demostración:
$\Rightarrow$) Si todo modelo de $\alpha_1, ..., \alpha_n$ ha de ser modelo de β, no es posible que haya un modelo de $\alpha_1, ..., \alpha_n$ que sea modelo de $\neg\beta$.
$\Leftarrow$) Si $(\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \neg\beta)$ no tiene modelo, entonces cualquier modelo de $\alpha_1, ..., \alpha_n$, será modelo de $(\alpha_1 \wedge \ldots \wedge \alpha_n)$ y también de β, pues no lo será de $\neg\beta$. □

Así pues, para probar que β es consecuencia lógica de $\Sigma = \{\alpha_1, ..., \alpha_n\}$, es suficiente probar que el conjunto de enunciados $\{\alpha_1, \ldots, \alpha_n, \neg\beta\}$ no tiene modelos, lo cual es equivalente a que el enunciado $(\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \neg\beta)$ no tenga modelos o que sea inconsistente.
Sea ψ una fórmula cualquiera. Por ejemplo $\psi = (\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \neg\beta)$. Sea φ la forma normal conjuntiva de la forma normal de Skolem de ψ[1]. Es un resultado de la lógica matemática que siempre es posible transformar de manera mecánica una fórmula cualquiera ψ en una fórmula en forma normal conjuntiva φ donde además los cuantificadores existenciales han sido eliminados por medio de constantes o funciones de Skolem, y donde los cuantificadores universales están presupuestos aunque no se escriben [15, 16]. Así, tal φ es una conjunción de disyunciones de fórmulas atómicas o negaciones de fórmulas atómicas. A esta forma se le llama forma clausular y la denotaremos con $Cl(\psi)$.
Cada una de las disyunciones de una forma clausular se llama *cláusula*; es decir, una *cláusula* es una disyunción de fórmulas tal que cada una de esas fórmulas es atómica o negación de atómica.

Teorema 1.2 *(Skolem)*
Sea ψ un enunciado y sea $Cl\,(\psi)$ la conjunción de las cláusulas correspondientes a ψ (o forma clausular de ψ). Entonces:

$$\psi \text{ tiene modelo } \Leftrightarrow Cl(\psi) \text{ tiene modelo}$$

Demostración: véase [9]

Definición 1.1 Dos fórmulas α, β son *lógicamente equiposibles* cuando una tiene un modelo si y sólo si la otra tiene un modelo. Es decir, las dos tienen o las dos no tienen modelos. La equiposibilidad lógica de dos fórmulas se denotará $\alpha \sim_{sat} \beta$.

[1]Estas son fórmulas con características especiales, de las que hablaremos más adelante.

En el caso del teorema, decimos que ψ y $Cl\,(\psi)$ son lógicamente equiposibles, i.e., $\psi \sim_{sat} Cl(\psi)$.
Obsérvese que la relación de equiposibilidad lógica es relación de equivalencia y que si dos fórmulas α y β, son lógicamente equivalentes ($\alpha \equiv \beta$), entonces son lógicamente equiposibles ($\alpha \sim_{sat} \beta$), pero no a la inversa. Por ejemplo $\exists x Px$ y Pc son lógicamente equiposibles pero no son lógicamente equivalentes [1]. Un caso particular es el siguiente:

Si $\psi = (\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \neg\beta)$, entonces: $\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \neg\beta$ no tiene modelos si y sólo si $Cl(\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \neg\beta)$ no tiene modelos.

Definición 1.2 Una fórmula está *rectificada* si:

1. No tiene presencias libres y acotadas de una misma variable, ni cuantificadores con alcances ajenos con la misma variable.
2. No tiene dos cuantificadores que acotan presencias de una misma variable (cuantificaciones dobles).
3. No tiene cuantificadores que no cuantifican (cuantificadores vacuos).

El siguiente es un algoritmo para transformar una fórmula cualquiera en un lenguaje de primer orden a su forma clausular:

1. Rectificar la fórmula.
2. Eliminar las implicaciones y las dobles implicaciones de la fórmula:
$$(\alpha \rightarrow \beta) \equiv \neg\alpha \vee \beta,\ (\alpha \leftrightarrow \beta) \equiv (\neg\alpha \vee \beta) \wedge (\neg\beta \vee \alpha)$$
3. Introducir al máximo las negaciones de la fórmula:
$$\neg\neg\alpha \equiv \alpha, \neg\,(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta, \neg\,(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$$
$$\neg\forall x\alpha \equiv \exists x\neg\alpha, \neg\exists x\alpha \equiv \forall x\neg\alpha$$
4. Skolemizar, es decir, eliminar los cuantificadores existenciales, de izquierda a derecha (se obtiene una fórmula lógicamente equiposible). Por ejemplo la skolemización de $\forall x\exists y Pxy$ es $\forall x Pxf(x)$ y la skolemización de $\exists y\neg Qy$ es $\neg Qc$, véase [18]
5. Presuponer los cuantificadores universales: todas las variables (libres) se consideran universalmente cuantificadas.
6. Pasar a la forma normal conjuntiva:
$$\alpha \vee (\beta \wedge \delta) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \delta)$$

El resultado de aplicar este algoritmo a una fórmula es la forma clausular de la misma. Como hemos visto, esta forma clausular es una conjunción de disyunciones de fórmulas atómicas o negaciones de atómicas (literales). La fórmula original y la forma clausular son lógicamente equiposibles:

$$\psi \sim_{sat} Cl(\psi)$$

Ejemplo 1.1 Sea $\psi \equiv \exists x \forall y \neg Pxy \rightarrow (\neg \forall y Qy \ \wedge \ \exists x Pax)$, entonces:

$$\begin{array}{lll} \psi & \equiv & \exists x \forall y \neg Pxy \rightarrow (\neg \forall w Qw \ \wedge \ \exists z Paz) \quad \text{Por 1.} \\ & \equiv & \neg \exists x \forall y \neg Pxy \vee (\neg \forall w Qw \ \wedge \ \exists z Paz) \quad \text{Por 2.} \\ & \equiv & \forall x \exists y Pxy \vee (\exists w \neg Qw \ \wedge \ \exists z Paz) \quad \text{Por 3.} \\ & \sim_{sat} & \forall x Pxf(x) \vee (\neg Qc \ \wedge \ Pad) \quad \text{Por 4.} \\ & \equiv & Pxf(x) \vee (\neg Qc \ \wedge \ Pad) \quad \text{Por 5.} \\ & \equiv & (Pxf(x) \vee \neg Qc) \ \wedge \ (Pxf(x) \vee Pad) \quad \text{Por 6.} \end{array}$$

y esta última fórmula es $Cl(\psi)$.

Hasta este punto los dos procedimientos motivo de esta exposición coinciden. A continuación presentaremos, partiendo de la forma clausular, un procedimiento basado en el Teorema de Herbrand en la sección 2 y un procedimiento basado en Resolución en la sección 3. En ambos casos se trabaja con clásulas, las cuales tienen variables y constantes pero no cuantificadores.

2 Procedimiento Basado en el Teorema de Herbrand

Teorema 2.1 *(Herbrand). Sea φ una fórmula sin cuantificadores con variables libres $x_1, x_2, \ldots, x_n$. Entonces: $\forall x_1 \forall x_2 \ldots \forall x_n \varphi$ no tiene modelo si y sólo si hay una sucesión finita $\varphi_1, \ldots, \varphi_k$ de instancias de φ que no tienen variables (es decir, con términos sin variables del lenguaje de φ) tal que $(\varphi_1 \wedge \ldots \wedge \varphi_k)$ no tiene modelo.*

Demostración: véase [9]

Un caso particular es aquel en que φ es la forma clausular:

$$\varphi = Cl\,(\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \neg\beta)$$

En este caso, φ es una conjunción de cláusulas por lo que no tiene cuantificadores. Retomemos la pregunta inicial y apliquemos los teoremas ya vistos.

$$\text{¿ Es } \beta \text{ una consecuencia lógica de } \alpha_1, \ldots, \alpha_n?, \text{ ¿ } \alpha_1, ..., \alpha_n \models \beta \text{ ?}$$

Tenemos las siguientes equivalencias:

$$\alpha_1, \ldots, \alpha_n \models \beta \underset{Teo.Básico}{\Leftrightarrow} (\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \neg\beta) \text{ no tiene modelo}$$

$$\underset{Teo.Skolem}{\Leftrightarrow} \varphi = Cl(\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \neg\beta) \text{ no tiene modelo}$$

$$\underset{Teo.Herbrand}{\Leftrightarrow} \varphi_1 \wedge \ldots \wedge \varphi_k \text{ no tiene modelo}$$

donde $\varphi_1, \ldots, \varphi_k$ es una sucesión finita de instancias de φ, sin variables
(es decir, en las que las variables de φ se han reemplazado por términos sin variables del lenguaje de φ).

El procedimiento que veremos genera instancias de φ, sin variables y para cada sucesión de instancias, hay que verificar si la conjunción de ellas no tiene modelo. Esto último es efectivamente decidible mediante un algoritmo [2, 13]:

- Si hay igualdades: Algoritmo de clases de equivalencia + Algoritmo de Davis-Putnam.
- Si no hay igualdades: Algoritmo de Davis-Putnam.

Así pues, tenemos lo siguiente:

- En caso de que β sea un teorema a partir de $\alpha_1, \ldots, \alpha_n$, entonces necesariamente existirán instancias de la fórmula φ correspondiente, tales que su conjunción no tendrá modelos y el procedimiento indicará que β es un teorema a partir de $\alpha_1, \ldots, \alpha_n$.
- En caso de que el procedimiento indique que β es un teorema a partir de $\alpha_1, \ldots, \alpha_n$, esto será porque se obtuvo una sucesión $\varphi_1, \ldots, \varphi_k$ de instancias de la fórmula φ correspondiente tales que el algoritmo nos dice que $\varphi_1 \wedge \ldots \wedge \varphi_k$ no tiene modelo y, por los teoremas vistos, tampoco φ tiene modelo y β sí es teorema a partir de $\alpha_1, \ldots, \alpha_n$.

Podemos concluir el siguiente metateorema:

β es un teorema a partir de $\alpha_1, \ldots, \alpha_n$ si y sólo si el procedimiento así lo indica.

Desde luego, cuando β no es un teorema a partir de $\alpha_1, \ldots, \alpha_n$, el procedimiento nada dice, pues tampoco dice que no lo sea. La idea es poner a prueba sólo enunciados que se cree que son teoremas. Parte de este procedimiento se ha implementado en el lenguaje Prolog [15] y en el lenguaje C [13].

2.1 Algoritmo de Clases de Equivalencia

Entrada: Un conjunto I de igualdades de términos

$$I = \{t_1 \approx t'_1, \ldots, t_n \approx t'_n\}$$

1. Cada término t que figura en una igualdad de I, forma una clase unitaria, y cada subtérmino que aparece en una igualdad de I, forma una clase unitaria.
2. Si $(t_i \approx t_j) \in I$, entonces se une la clase de t_i con la clase de t_j.

3. Si $f(t_1, \ldots, t_n)$ y $f(s_1, \ldots, s_n)$ son dos términos (el símbolo "f" es común) que pertenecen a ciertas clases y si para todo i ; $(i = 1, \ldots, n)$ resulta que t_i y s_i pertenecen a la misma clase, entonces se une la clase de $f(t_1, \ldots, t_n)$ con la clase de $f(s_1, \ldots, s_n)$.

4. Se repite el paso 3 cuantas veces sea necesario, es decir, hasta que al revisar todos los términos no se produce ninguna nueva unión de clases.

Salida: Un conjunto de clases de equivalencia (que son las igualdades determinadas por I). Decimos que todos los términos de cada clase son iguales y tomamos uno de cada clase como su representante.

Ejemplo 2.1 (Ejemplo del Algoritmo de Clases de Equivalencia para igualdades).

$$I = \{f(f(b,a), g(a)) \approx f(b, f(a, g(a))),\ f(b,e) \approx b,\ f(a, g(a)) \approx e,\\ f(b,a) \approx f(c,a),\ f(c,e) \approx c,\ f(f(c,a), g(a)) \approx f(c, f(a, g(a)))\}$$

Observación: En I hay seis igualdades y aparecen doce términos, pero al considerar todos los subtérminos son 14.
Las constantes son: a, b, c, e.
Los símbolos de función son f (de dos argumentos) y g (de un argumento) Los subtérminos nuevos son dos: "$g(a)$" y "a".
Paso 1: 14 clases unitarias. Después de los pasos 1 y 2 hay 8 clases:

$$\{f(f(b,a), g(a)), f(b, f(a, g(a)))\}\ \{f(b,e), b\}\ \{f(a, g(a)), e\}\\ \{f(b,a), f(c,a)\}\ \{f(c,e), c\}\\ \{f(f(c,a), g(a)), f(c, f(a, g(a)))\}\ \{g(a)\}\ \{a\}$$

Después del paso 3 hay 5 clases:

$$\{f(c,e), c, f(f(c,a), g(a)), f(c, f(a, g(a))), f(f(b,a), g(a)),\\ f(b, f(a, g(a))), f(b,e), b\}\ \{f(a, g(a)), e\},\ \{f(b,a), f(c,a)\}\ \{g(a)\}\ \{a\}$$

Observación: Como c y b están en la misma clase, se tiene que son el mismo. Sólo se toma a uno de ellos como representante (es decir, son sustituibles). En caso de tener un predicado o negación de igualdad con b, podemos poner c en su lugar.

Todo lo anterior se verá más claro con varios ejemplos en los que aplicaremos el procedimiento. En cada caso conviene recordar cómo se justifica cada paso.

Ejemplo 2.2 "La reflexividad de una relación" es consecuencia lógica de que "la relación sea simétrica, transitiva, y sin puntos aislados". Es decir:

$$\forall xy\,(Pxy \to Pyx), \forall xyz\,(Pxy \wedge Pyz \to Pxz), \forall x \exists y\,(Pxy \vee Pyx)\\ \models \forall x Pxx$$

$$\Leftrightarrow \forall x, y\,(Pxy \to Pyx) \wedge \forall x, y, z\,(Pxy \wedge Pyz \to Pxz) \wedge\\ \forall x \exists y\,(Pxy \vee Pyx) \wedge\ \neg \forall x Pxx \text{ no tiene modelo}$$

$$\Leftrightarrow (\neg Pxy \vee Pyx) \wedge (\neg Pxy \vee \neg Pyz \vee Pxz) \wedge (Pxf(x) \vee Pf(x)x) \wedge (\neg Pcc) \text{ no tiene modelo}$$

La fórmula a instanciar es:

$$(\neg Px_1x_2 \vee Px_2x_1) \wedge (\neg Px_3x_4 \vee \neg Px_4x_5 \vee Px_3x_5) \wedge (Px_6f(x_6) \vee Pf(x_6)x_6) \wedge (\neg Pcc)$$

Constantes: c
Funciones: f (de un argumento)
Instancia 1: $x_1, x_4/f(c) \quad x_2, x_3, x_5, x_6/c$
Instancia 2: $x_1/c \quad x_2/fc$

Instancia 1	*Forma*
$\neg Pf(c)c \vee Pcf(c)$	$\neg A \vee B$
$\neg Pcf(c) \vee \neg Pf(c)c \vee Pcc$	$\neg B \vee \neg A \vee C$
$Pcf(c) \vee Pf(c)c$	$B \vee A$
$\neg Pcc$	$\neg C$

Instancia 2	*Forma*
$\neg Pcf(c) \vee Pf(c)c$	$\neg B \vee A$

Forma $\rightarrow$ Algoritmo Davis-Putnam[2] $\rightarrow$ No tiene modelo.
$\therefore$ Por el teorema de Herbrand, el teorema de Skolem y el teorema Básico, la consecuencia lógica es cierta.

Ejemplo 2.3 Supongamos que: "cualquier individuo es inocente o culpable pero no ambas cosas", "Si Rafael es culpable, entonces Javier y María son culpables", "Pablo es inocente si Rafael lo es", "Javier es inocente, pero al menos uno de los otros tres es culpable". La pregunta es:
¿ Quién es culpable?, ¿ Se infiere lógicamente que María es culpable?

$$\forall x\left[(Ix \vee Cx) \wedge \neg(Ix \wedge Cx)\right], (Cr \rightarrow Cj \wedge Cm), (Ir \rightarrow Ip), [Ij \wedge (Cr \vee Cm \vee Cp)] \models Cm$$

$$\Leftrightarrow (Ix \vee Cx) \wedge (\neg Ix \vee \neg Cx) \wedge (\neg Cr \vee (Cj \wedge Cm)) \wedge (\neg Ir \vee Ip) \wedge Ij \wedge (Cr \vee Cm \vee Cp) \wedge \neg Cm \text{ no tiene modelo}$$

$$\Leftrightarrow (Ix_1 \vee Cx_1) \wedge (\neg Ix_2 \vee \neg Cx_2) \wedge (\neg Cr \vee Cj) \wedge (\neg Cr \vee Cm) \wedge (\neg Ir \vee Ip) \wedge Ij \wedge (Cr \vee Cm \vee Cp) \wedge \neg Cm \text{ no tiene modelo}$$

Constantes: m, r, p, j
Funciones: No hay.
Instancia 1: $x_1, x_2/m$
Instancia 2: $x_1, x_2/r$
Instancia 3: $x_1, x_2/p$

[2]El algoritmo de Davis-Putnam es un método que decide si un conjunto de instancias de cláusulas sin variables tiene o no un modelo (Cf. sec. 2.2).

$Instancia\ 1$	$Forma$
$Im \vee Cm$	$A \vee B$
$\neg Im \vee \neg Cm$	$\neg A \vee \neg B$
$\neg Cr \vee Cj$	$\neg C \vee D$
$\neg Cr \vee Cm$	$\neg C \vee B$
$\neg Ir \vee Ip$	$\neg E \vee F$
Ij	G
$Cr \vee Cm \vee Cp$	$C \vee B \vee H$
$\neg Cm$	$\neg B$

$Instancia\ 2$	$Forma$
$Ir \vee Cr$	$E \vee C$
$\neg Ir \vee \neg Cr$	$\neg E \vee \neg C$

$Instancia\ 3$	$Forma$
$Ip \vee Cp$	$F \vee H$
$\neg Ip \vee \neg Cp$	$\neg F \vee \neg H$

Forma→ Algoritmo Davis-Putnam → No tiene Modelo
$\therefore$ se infiere lógicamente que María es culpable.

Ejemplo 2.4 Si "todos los barberos de Oaxtepec rasuran única y exclusivamente a los padres de todos los hombres que no rasuran a su propio padre y sólo a esos", entonces se infiere lógicamente que "No hay barberos en Oaxtepec". Es decir:

$$\begin{aligned}
&\forall x\left[Bx \rightarrow \forall y\left(Rxf\left(y\right) \leftrightarrow \neg Ryf\left(y\right)\right)\right] \models \neg\exists xBx \\
&\Leftrightarrow \forall x\left[Bx \rightarrow \forall y\left(Rxf\left(y\right) \leftrightarrow \neg Ryf\left(y\right)\right)\right] \wedge \exists xBx \text{ no tiene modelo} \\
&\Leftrightarrow \left[\neg Bx \vee \left[\left(\neg Rxf\left(y\right) \vee \neg Ryf\left(y\right)\right) \wedge \left(Ryf\left(y\right) \vee Rxf\left(y\right)\right)\right]\right] \\
&\quad \wedge Bc \text{ no tiene modelo} \\
&\Leftrightarrow \left[\neg Bx \vee \neg Rxf\left(y\right) \vee \neg Ryf\left(y\right)\right] \wedge \left[\neg Bx \vee Ryf\left(y\right) \vee Rxf\left(y\right)\right] \\
&\quad \wedge Bc \text{ no tiene modelo} \\
&\Leftrightarrow \left[\neg Bx_1 \vee \neg Rx_1f\left(x_2\right) \vee \neg Rx_2f\left(x_2\right)\right] \wedge \left[\neg Bx_3 \vee Rx_4f\left(x_4\right) \vee\right. \\
&\quad \left.Rx_3f\left(x_4\right)\right] \wedge\ Bc \text{ no tiene modelo}
\end{aligned}$$

Constantes: c
Funciones: f
Predicados: B, R
Instancia 1: $x_1, x_2, x_3, x_4/c$

$Instancia\ 1$	$Forma$
$\neg Bc \vee \neg Rcf\left(c\right) \vee \neg Rcf\left(c\right)$	$\neg B \vee \neg A \vee \neg A$
$\neg Bc \vee Rcf\left(c\right) \vee Rcf\left(c\right)$	$\neg B \vee A \vee A$
Bc	B

Forma → Algoritmo Davis-Putnam → No tiene Modelo

Ejemplo 2.5 Si se suponen los "axiomas de grupo", se puede deducir lógicamente la "ley de cancelación".

$$\begin{array}{l}\forall x,y,z\left[ff\left(xy\right)z\approx fxf\left(yz\right)\right],\forall wf\left(we\right)\approx w,\forall u\exists vf\left(uv\right)\approx e\\ \models\forall x,y,z\left(f\left(yx\right)\approx f\left(zx\right)\rightarrow y\approx z\right)\\ \Leftrightarrow\left[f\left(f\left(xy\right)z\right)\approx fxf\left(yz\right)\right]\wedge\left[f\left(we\right)\approx w\right]\wedge\left[f\left(u,g\left(u\right)\right)\approx e\right]\\ \quad\wedge\left[fba\approx fca\wedge\neg b\approx c\right]\text{ no tiene modelo}\\ \Leftrightarrow\left[f\left(f\left(x_1x_2\right)x_3\right)\approx f\left(x_1f\left(x_2x_3\right)\right)\right]\wedge\left[f\left(x_4e\right)\approx x_4\right]\wedge\\ \quad\left[f\left(x_5,g\left(x_5\right)\right)\approx e\right]\wedge\left[fba\approx fca\wedge\neg b\approx c\right]\text{ no tiene modelo.}\end{array}$$

Constantes: a,b,c,e

Funciones: f (de dos argumentos), g (de un argumento)
Instancia 1: $x_1,x_4/b$, $x_2,x_5/a$, $x_3/g\left(a\right)$
Instancia 2: $x_1,x_4/c$,x_2/a, $x_3/g\left(a\right)$

$$\begin{array}{c}Instancia\ 1\\ f\left(f\left(ba\right)g\left(a\right)\right)\approx f\left(bf\left(ag\left(a\right)\right)\right)\\ f\left(be\right)\approx b\\ f\left(ag\left(a\right)\right)\approx e\\ f\left(ba\right)\approx f\left(ca\right)\\ \neg b\approx c\end{array}$$

$$\begin{array}{c}Instancia\ 2\\ f\left(f\left(ca\right)g\left(a\right)\right)\approx f\left(cf\left(ag\left(a\right)\right)\right)\\ f\left(ce\right)\approx c\end{array}$$

Aplicando el algoritmo de clases de equivalencias para igualdades, (véase el ejemplo del algoritmo) se obtiene que $''b''$ y $''c''$ son el mismo y por otro lado tenemos $\neg b\approx c$ (desigualdad imposible), por tanto: No tiene Modelo y esto prueba la consecuencia lógica.

Ejemplo 2.6 Si "Una operación binaria cerrada en un álgebra es asociativa y hay soluciones izquierda y derecha x,y para cualesquiera ecuaciones $xa=b$ y $ay=b$ donde a y b son elementos cualesquiera del álgebra", entonces se infiere lógicamente que "hay un elemento neutro derecho", es decir, si $Pxyz$ representa a $xy=z$ entonces:

$$\begin{array}{c}\forall x\forall y\forall z\forall u\forall v\forall w\left[\left(Pxyu\ \wedge\ Pyzv\ \wedge\ Pxvw\rightarrow Puzw\right)\wedge\right.\\ \left.\left(Pxyu\ \wedge\ Pyzv\ \wedge\ Puzw\rightarrow Pxvw\right)\right],\ \forall x\forall y\exists zPzxy,\ \forall x\forall y\exists zPxzy,\\ \forall x\forall y\exists zPxyz\models\exists x\forall yPyxy\end{array}$$

tenemos

$$\begin{array}{rl}\psi= & \forall x\forall y\forall z\forall u\forall v\forall w\left[\left(Pxyu\ \wedge\ Pyzv\ \wedge\ Pxvw\rightarrow Puzw\right)\right.\\ & \left.\wedge\left(Pxyu\ \wedge\ Pyzv\ \wedge\ Puzw\rightarrow Pxvw\right)\right]\ \wedge\ \forall x\forall y\exists zPzxy\wedge\\ & \forall x\forall y\exists zPxzy\ \wedge\ \forall x\forall y\exists zPxyz\ \wedge\ \forall x\exists y\neg Pyxy\end{array}$$

ahora escribimos las cláusulas de $Cl(\psi)$:

1. $\neg Px_1y_1u_1 \vee \neg Py_1z_1v_1 \vee \neg Px_1v_1w_1 \vee Pu_1z_1w_1$
2. $\neg Px_2y_2u_2 \vee \neg Py_2z_2v_2 \vee \neg Pu_2z_2w_2 \vee Px_2v_2w_2$
3. $Pg(x_3, y_3)x_3y_3$
4. $Px_4h(x_4, y_4)y_4$
5. $Px_5y_5f(x_5, y_5)$
6. $\neg Pk(x_6)x_6k(x_6)$

Constantes: No hay constantes en el lenguaje de $Cl(\psi)$, en este caso se introduce una nueva constante a. Funciones: g, h, f (de dos argumentos), k (de un argumento). Predicados: P (de tres argumentos).

Haciendo la sustitución $x_1/g(a, k(h(a, a))),\ y_1/a,\ u_1/k(h(a, a))$ $z_1/h(a, a),\ v_1/a,\ w_1/k(h(a, a)),\ x_3/a,\ y_3/k(h(a, a)),\ x_4/a,\ y_4/a,$ $x_6/h(a, a)$, sin considerar a las cláusulas 2 y 5, se obtiene

1'. $\neg Pg(a, k(h(a, a))ak(h(a, a))) \vee \neg Pah(a, a)a$
$\vee \neg Pg(a, k(h(a, a)))ak(h(a, a)) \vee Pk(h(a, a))h(a, a)k(h(a, a))$

3'. $Pg(a, k(h(a, a)))ak(h(a, a))$

4'. $Pah(a, a)a$

6'. $\neg Pk(h(a, a))h(a, a)k(h(a, a))$

Para este arreglo de cláusulas instanciadas el algoritmo de Davis-Putnam reporta que no tiene modelo. Se puede ver que efectivamente no es proposicionalmente satisfacible, pues es de la forma

$$(\neg A \vee \neg B \vee \neg A \vee C) \ \wedge \ A \ \wedge \ B \ \wedge \ \neg C$$

así que la consecuencia lógica es cierta.

2.2 El Algoritmo de Davis-Putnam

Definición 2.1 Si L es una literal (una fórmula atómica P o la negación $\neg P$ de una fórmula atómica), definimos la *literal contraria* L^c como:

$$L^c = \begin{cases} \neg P & si\ L = P \\ P & si\ L = \neg P \end{cases}$$

Entrada: Un conjunto S de cláusulas instanciadas.

1. Retirar todas las cláusulas que sean tautologías. El conjunto obtenido será el nuevo conjunto S.

$$\alpha \wedge (\gamma \vee L \vee L^c) \equiv \alpha$$

2. Iterar las reglas que siguen hasta que S sea explícitamente contradictorio o vacío. En el primer caso S no tiene modelo, en el segundo caso sí tiene modelo.

   (a) Si hay una cláusula con una sola literal L, retirar todas las cláusulas donde figure L (incluyendo a L) y remover todas las presencias de L^c:

   $$L \wedge (L \vee \alpha) \wedge (L^c \vee \beta) \wedge \gamma \sim_{sat} \beta \wedge \gamma$$

   (b) Si L figura en alguna cláusula pero no hay presencias de L^c en ninguna otra cláusula, retirar todas las cláusulas donde L figura:

   $$(L \vee \alpha) \wedge \beta \sim_{sat} \beta$$

   (c) Remover toda cláusula que contenga a otra cláusula. Si la cláusula A esta incluida o forma parte de la cláusula B entonces:

   $$A \wedge B \wedge \gamma \equiv A \wedge \gamma$$

   (d) Si las reglas anteriores no son aplicables, elegir una literal L tal que L y L^c estén en el conjunto S y remplazar el conjunto S por los dos conjuntos de cláusulas siguientes: el conjunto de cláusulas S_1 que resulta de eliminar de S todas las cláusulas que contengan a L y borrar todas las presencias de L^c, y el conjunto S_2 que resulta de eliminar de S todas las cláusulas que contengan a L^c y eliminar todas las presencias de L. En lo que sigue suponemos que en α, β, γ no figuran ni L ni L^c:

   $$\begin{array}{c}(L \vee \alpha) \wedge (L^c \vee \beta) \wedge \gamma \text{ no tiene modelo} \\ \Leftrightarrow \alpha \wedge \gamma \text{ no tiene modelo y } \beta \wedge \gamma \text{ no tiene modelo.}\end{array}$$

A continuación mostramos un par de ejemplos de aplicación del algoritmo de Davis-Putnam:

Ejemplo 2.7 Correspondiente al ejemplo 2.2.

$$\left|\begin{array}{c}\neg A \vee B \\ \neg B \vee \neg A \vee C \\ B \vee A \\ \neg C \\ \neg B \vee A\end{array}\right| \underset{(a)}{\rightarrow} \left|\begin{array}{c}\neg A \vee B \\ \neg B \vee \neg A \\ B \vee A \\ \neg B \vee A\end{array}\right| \underset{(d)}{\rightarrow} \left|\begin{array}{c}B \\ \neg B\end{array}\right| \quad \left|\begin{array}{c}B \\ \neg B\end{array}\right|$$

$\therefore$ El conjunto de entrada no tiene modelo

Ejemplo 2.8

$$\left|\begin{array}{c}Pc \vee Qa \vee \neg Rd \\ \neg Pc \vee \neg Rd \\ \neg Qa \\ Rd \vee Qa\end{array}\right| \underset{(a)}{\rightarrow} \left|\begin{array}{c}Pc \vee \neg Rd \\ \neg Pc \vee \neg Rd \\ Rd\end{array}\right| \underset{(a)}{\rightarrow} \left|\begin{array}{c}Pc \\ \neg Pc\end{array}\right|$$

$\therefore$ El conjunto de entrada no tiene modelo

2.3 Algunas ideas de Heurísticas para elegir sustituciones o instancias.

Idea: Sustituir variables por constantes que aparezcan en literales contrarias o iguales, de otra cláusula. Dada una variable x y un predicado P que la tiene, buscar entonces alguna constante $''c''$ (o término t), que figure en el predicado $\neg P$ de otra cláusula y si lo hay, sustituir c en vez de x (x/c), o t en vez de x (x/t).

$$\left|\begin{array}{c} Px \\ \neg Pc \end{array}\right| \quad \underset{x/c}{\rightarrow} \quad \left|\begin{array}{c} Pc \\ \neg Pc \end{array}\right|$$

En breve:

H1: Dada una variable x, fijarse en las literales L_i que tienen a x; luego fijarse en las constantes c_j que aparecen en literales contrarias L_i^c de otra cláusula y sustituir x/c_j.

H2: Dada una variable x, fijarse en la literal que tiene x, y si hay otra presencia (en otra cláusula) de la misma literal con una constante c (en el lugar de la x), entonces sustituir x/c.

Ejemplo 2.9 Véase junto con el ejemplo 2.8.

$$\left|\begin{array}{c} Pc \vee Qy \vee \neg Rw \\ \neg Px \vee \neg Rd \\ \neg Qa \\ Rz \vee Qy \end{array}\right| \quad \overrightarrow{x/c, w/d, y/a, z/d} \quad \left|\begin{array}{c} Pc \vee Qa \vee \neg Rd \\ \neg Pc \vee \neg Rd \\ \neg Qa \\ Rd \vee Qa \end{array}\right|$$

Hay muchos demostradores automáticos basados en ideas muy diversas, pero creemos que muy pocos podrían ser semejantes o ayudar a este procedimiento. En particular, es de nuestro interés el demostrador NTAB para el algoritmo de Davis-Putnam (implementado en ambiente UNIX). Nuestro interés es trabajar con ellos y saber si en nuestro medio ya se le conoce, así como saber si hay algún intento en la dirección mostrada en este procedimiento.

3 Procedimiento Basado en Resolución

El segundo método de demostración automática que presentamos en este trabajo es el método clásico de resolución desarrollado originalmente por Robinson en 1965 [14]. La regla de resolución binaria juega un papel fundamental en la programación lógica [6], siendo la base del lenguaje de programación Prolog, se trata de una generalización de

varias reglas de inferencia conocidas, tales como:

$$\frac{A \quad A \to C}{C} \qquad \frac{A \vee B \quad \neg A}{B} \qquad \frac{A \vee B \quad A \to C}{B \vee C}$$

$$\frac{A \quad \neg A \vee C}{C} \qquad \frac{A \vee B \quad \neg A \vee \neg C}{B \vee \neg C} \qquad \frac{A \to B \quad B \to C}{A \to C}$$

Definición 3.1 Sea W un conjunto no vacío de fórmulas o de términos. Una sustitución σ se llama un *unificador* de W si $\mid W\sigma \mid = 1$, es decir, si la imagen de W bajo σ tiene un solo elemento.

Ejemplo 3.1 Sea $W = \{P(x, f(y)), P(x, f(x)), P(u, v)\}$ entonces la sustitución $\sigma = \{x/a, y/a, u/a, v/f(a)\}$ es un unificador de W, puesto que $W\sigma = \{P(a, f(a))\}$

Un conjunto de fórmulas puede tener una infinidad de unificadores o ninguno. Dado un conjunto finito de fórmulas T, siempre es decidible mediante un algoritmo [18, 5, 6] si T es unificable o no; además, si T es unificable, el algoritmo proporciona un unificador llamado *unificador más general*.

Definición 3.2 Un unificador σ de un conjunto de fórmulas o términos W, se llama *unificador más general* (abreviado umg) si para todo unificador τ de W, existe una sustitución ϑ, tal que $\sigma\vartheta = \tau$.

Ejemplo 3.2 Sean $W = \{f(g(a, x), g(y, b)), f(z, g(u, v))\}$, $\tau = \{x/a, z/g(a, a), y/u, v/b\}, \sigma = \{z/g(a, x), y/u, v/b\}$. Entonces τ y σ son unificadores de W y σ resulta ser un umg; en particular si $\vartheta = \{x/a\}$ entonces $\sigma\vartheta = \tau$.

La regla de resolución binaria tiene como objetivo eliminar literales de cláusulas mediante un umg.

Definición 3.3 Sean $C_1 = L \vee Q_1 \vee \ldots \vee Q_m, C_2 = \neg M \vee R_1 \vee \ldots \vee R_n$ dos cláusulas que no tienen variables en común. Sea σ un umg de $\{L, M\}$, el siguiente esquema se conoce como regla de *resolución binaria*:

$$\frac{L \vee Q_1 \vee \ldots \vee Q_m \qquad \neg M \vee R_1 \vee \ldots \vee R_n}{(Q_1 \vee \ldots \vee Q_m \vee R_1 \vee \ldots \vee R_n)\sigma}$$

La cláusula $(Q_1 \vee \ldots \vee Q_m \vee R_1 \vee \ldots \vee R_n)\sigma$ se llama *resolvente* de C_1 y C_2, en el caso particular en que $C_1 = L$ y $C_2 = \neg M$, la cláusula resolvente se llama *cláusula vacía* y se denota $\square$.

Ejemplo 3.3 Sean $C_1 = P(x, f(x)) \vee \neg Q(x, y) \vee \neg R(y)$, $C_2 = \neg P(a, z) \vee \neg R(z)$ entonces la cláusula $\neg Q(a, y) \vee \neg R(y) \vee \neg R(f(a))$ es un resolvente para C_1 y C_2, es decir, la instancia

$$\frac{P(x, f(x)) \vee \neg Q(x, y) \vee \neg R(y) \qquad \neg P(a, z) \vee \neg R(z)}{\neg Q(a, y) \vee \neg R(y) \vee \neg R(f(a)}$$

de resolución binaria es válida, siendo $\sigma = \{x/a, z/f(a)\}$ el umg.

La regla de resolución binaria es correcta, es decir, al aplicarla el resolvente es consecuencia lógica de las premisas; si bien no hay garantía de derivar un teorema directamente, resolución (bajo ciertas hipótesis) es refutación-completa: si se tiene un conjunto inconsistente de cláusulas, necesariamente se llegará a $\square$, como lo asegura el siguiente teorema.

Teorema 3.1 *(Loveland)*
Si K es un conjunto de cláusulas entonces:

$$K \text{ es inconsistente} \Leftrightarrow \square \text{ se deduce de } K \text{ usando resolución}$$

Demostración: véase [7].

El procedimiento de demostración automática en este caso consiste en encontrar una deducción de la cláusula vacía. Si tenemos un teorema (o una pregunta que tiene respuesta) P, entonces para probar P por refutación dado $\{C_0, C_1, \ldots, C_n\}$ hay que proceder como sigue:

1. Negar P.
2. Construir $K = \{\neg P, C_0, C_1, \ldots, C_n\}$ en forma clausular.
3. Aplicar pasos de resolución a partir de K hasta hallar $\square$.

Esto está justificado por los teoremas 1.1., 1.2 y 3.1.
Más aún, se pueden obtener respuestas componiendo substituciones a lo largo de una rama del árbol de prueba que lleva a $\square$. El principal problema consiste en reducir el factor de "ramaje" del árbol, para lo cual se han implementado reglas derivadas de resolución binaria como lo es la hiperresolución (ver la sección 4).

Veamos como se resuelve el ejemplo 2.2 mediante resolución

Ejemplo 3.4 La reflexividad de una relación es consecuencia lógica de la simetría, transitividad y la ausencia de puntos aislados.
Las cláusulas con variables renombradas son: $\neg Px_1x_2 \vee Px_2x_1$, $\neg Px_3x_4 \vee \neg Px_4x_5 \vee Px_3x_5$, $Px_6f(x_6) \vee Pf(x_6)x_6$, $\neg Pcc$

$$\frac{\begin{array}{l}\neg Px_1x_2 \vee Px_2x_1 \\ Px_6f(x_6) \vee Pf(x_6)x_6\end{array}}{Pf(c)c \vee Pf(c)c} \quad \sigma_1 = \{x_1/c, x_2/f(c), x_6/c\}$$

Como $Pf(c)c \vee Pf(c)c \equiv Pf(c)c$ tenemos

$$\frac{\begin{array}{c} Pf(c)c \\ \neg Px_1x_2 \vee Px_2x_1 \end{array}}{Pcf(c)} \qquad \sigma_2 = \{x_1/f(c), x_2/c\}$$

$$\frac{\begin{array}{c} Pcf(c) \\ \neg Px_3x_4 \vee \neg Px_4x_5 \vee Px_3x_5 \end{array}}{\neg Pf(c)x_5 \vee Pcx_5} \qquad \sigma_3 = \{x_3/c, x_4/f(c)\}$$

$$\frac{\begin{array}{c} \neg Pf(c)x_5 \vee Pcx_5 \\ \neg Pcc \end{array}}{\neg Pf(c)c} \qquad \sigma_4 = \{x_5/c\}$$

$$\frac{\begin{array}{c} \neg Pf(c)c \\ Pf(c)c \end{array}}{\square} \qquad \sigma_5 = \emptyset$$

4 La Nutria

Presentamos un ejemplo de un demostrador automático que implementa al método de resolución para lógica de primer orden con igualdad. Se le denomina la nutria (Otter en inglés) y fue desarrollado en el Laboratorio Nacional de Argonne, USA. Las siglas "otter" significan **O**rganized **T**echniques for **T**heorem-proving and **E**ffective **R**esearch. Las consideraciones primarias para su diseño fueron el desarrollo, la portabilidad y la extensibilidad. Otter está implementado en C para ambiente Unix, aunque también está disponible para Windows y Macintosh de manera gratuita, incluyendo el codigo fuente en:

http://www.mcs.anl.gov/home/mccune/ar/otter

Si bien Otter tiene como regla de inferencia básica la resolución binaria, posee además otras reglas de inferencias derivadas de ésta, pero más poderosas, en el sentido de ahorro de tiempo y memoria, como lo son la hiperresolución, la UR-Resolución y la Paramodulación para la reescritura de términos. Algunos de los procesos que Otter tiene implementados son:

1. Conversión de fórmulas de primer orden a forma clausular.
2. Simplificación de Cláusulas.
3. Peso.
4. Funciones y Predicados Evaluables.

A continuación describiremos brevemente, mediante ejemplos, las reglas básicas de inferencia de Otter.

Definición 4.1 Una cláusula es una *cláusula unitaria* si contiene exactamente una literal.

El objetivo de la regla de UR-Resolución es obtener cláusulas unitarias a partir de un conjunto de cláusulas, una de las cuales debe ser no unitaria, mientras que las demás son unitarias. Veamos un ejemplo:

Ejemplo 4.1 Mediante UR-Resolución con el unificador
$\mu = \{x/Tania, y/Pablo, z/Susana\}$ y a partir de las cláusulas:

$$\begin{array}{l} \neg\, Casado(x,y) \ \vee\ \neg\, Madre(x,z) \ \vee\ Padre(y,z) \\ \quad Casado(Tania, Pablo) \\ \neg\, Padre(Pablo, Susana) \end{array}$$

se obtiene:

$$\neg\, Madre(Tania, Susana)$$

Una aplicación exitosa de UR-Resolución se puede ver como una sucesión de pasos de resolución binaria en la que cada uno de ellos involucra una cláusula unitaria.Todas las aplicaciones de resolución binaria deben darse simultáneamente de manera que no se produzcan cláusulas intermedias.

Pasemos a describir la regla de hiperresolución.

Definición 4.2 Una cláusula es *positiva* si todas sus literales son átomos no negados. Es *negativa* si todas sus literales son átomos negados y es *mixta* en otro caso.

El objetivo de la regla de hiperresolución es producir una cláusula positiva a partir de un conjunto de cláusulas, una de ellas negativa o mixta, y las demás positivas. Veamos un ejemplo

Ejemplo 4.2 Mediante Hiperresolución con el unificador
$\mu = \{x/Tania, y/Pablo, z/Susana\}$ y a partir del conjunto

$$\begin{array}{l} \neg\, Casado(x,y) \ \vee\ \neg\, Madre(x,z) \ \vee\ Padre(y,z) \\ \quad Casado(Tania, Pablo) \ \vee\ Masviejo(Tania, Pablo) \\ \quad Madre(Tania, Susana) \end{array}$$

obtenemos:

$$Padre(Pablo, Susana) \ \vee\ Masviejo(Tania, Pablo)$$

Otter utiliza la reescritura de términos mediante la regla de paramodulación, cuyo objetivo es la substitución de iguales de una cláusula a otra, requiriendo que al menos una de ellas contenga una literal de igualdad positiva, y produciendo una cláusula en la que la sustitución correspondiente a la literal de igualdad se ha realizado.

Ejemplo 4.3 Mediante paramodulación con $\mu = \{x/Jaime\}$ y a partir del conjunto

$$\begin{array}{l} Masviejo(padre(x), x) \\ Padre(Jaime) = Rodolfo \end{array}$$

se obtiene

$$Masviejo(Rodolfo, Jaime)$$

La regla de paramodulación incluye, la substitución ordinaria de iguales, aunque también produce cláusulas que no podrían obtenerse por substitución ordinaria. Para una exposición completa de estas reglas de inferencia ver [21].

4.1 El Proceso de Inferencia de Otter

Una vez realizada la transformación de fórmulas a cláusulas, Otter inicia el proceso de inferencia para tratar de llegar a la cláusula vacía $\square$. Dicho proceso se basa en la llamada estrategia del conjunto de soporte [22] y en un proceso de jerarquización de las cláusulas llamado *peso*, el cual asigna prioridades a términos. La noción esencial es circunscribir el conjunto de términos que puedan figurar en las cláusulas y limitar la complejidad de las cláusulas retenidas.
Para lograr esto se utiliza un mecanismo de estimación de la complejidad. El más común (y bastante útil) consiste en contar los símbolos que figuran en el término o cláusula. El mecanismo de inferencia básico es el algoritmo de la cláusula dada, el cual es una implementación de la estrategia del conjunto de soporte.

Definición 4.3 Una cláusula C tiene *soporte* si y sólo si C es una cláusula de entrada perteneciente al conjunto de soporte, o C se derivó de $R_1, \ldots, R_n$ donde al menos una R_i tiene soporte.

Como se ve, las cláusulas iniciales con soporte son aquellas seleccionadas por el usuario. Generalmente, el conjunto inicial de soporte consta de las cláusulas que describen al problema en cuestión junto con la negación de la conclusión del problema. Otter mantiene 4 listas de cláusulas, siendo las más relevantes:

1. usable: esta lista contiene las cláusulas que están disponibles para hacer inferencias.

2. cds: lista del conjunto de soporte. Las cláusulas de esta lista no están disponibles para hacer inferencias; estan esperando para participar en la búsqueda de la cláusula vacía.

El ciclo principal para inferir, procesar cláusulas y buscar una refutación opera principalmente en estas 2 listas.

Mientras (la lista cds sea no vacía y no se haya encontrado una refutación) hacer lo siguiente:

1. Asignar la cláusula más "ligera" de la lista cds a *claúsula-dada*.

2. Mover *cláusula-dada* de la lista cds a la lista usable.

3. Inferir y procesar cláusulas nuevas usando las reglas de inferencia establecidas; cada cláusula nueva debe tener a *cláusula-dada* como una de sus premisas y cláusulas de la lista usable como sus demás premisas; las cláusulas nuevas que pasan ciertas pruebas de retención se agregan a la lista cds.

Fin del Ciclo.

Para una descripción más completa y detallada del proceso de inferencia, así como de las pruebas de retención ver [10].
La efectividad de Otter es extraordinaria. Por ejemplo, cada uno de los ejemplos 2.2 a 2.6 de la sección anterior fueron resueltos, desplegando la demostración, en menos de 1 segundo, lo cual no es de asombrarse teniendo en cuenta que Otter ha resuelto problemas abiertos en diversas áreas como Teoría de Grupos, Geometría Algebraica, Álgebras Booleanas, Teoría de Retículos y Lógica Combinatoria. Véase la página web de Otter o [20].

5 Conclusiones

La demostración automática de teoremas en lógica de primer orden con igualdad que inició en la década de los años 70 [2, 4, 12, 14, 19] ha tenido logros que se pueden considerar impresionantes. Actualmente está en pleno desarrollo [5, 10, 20, 21]. Mediante la demostración automática se han obtenido pruebas de resultados que no eran conocidos en matemáticas, es decir, que no habían sido demostrados por métodos tradicionales. Algunos de ellos se han demostrado con Otter. En parte, se ha realizado la propuesta de Leibniz de un cálculo universal de razonamiento para llevar a cabo pruebas matemáticas en forma mecánica. Sin embargo, falta mucho por hacer, como la verificación de más algoritmos y estrategias más poderosas, la implementación de más heurísticas, la busqueda de reglas de inferencia más efectivas y generar de manera más eficiente modelos y contraejemplos más complejos.

Bibliografía

[1] J.A. Amor, J.A. Ramírez, Un Procedimiento de Prueba Automática para Enunciados Universalmente Válidos en interacción con el usuario, Memoria IX Reunión Nacional de Inteligencia Artificial, SMIA,1992.

[2] M. Davis H. Putnam, A Computing Procedure for Cuantification Theory, J.A.C.M, 1960.

[3] M. Fitting, First Order Logic and Automated Theorem Proving, Graduate Texts in Computer Science, Springer Verlag, 1996.

[4] P.C. Gilmore, A Proof Method for Cuantification Theory, IBM J. Res. Develop. **4**, 28-35, 1960.

[5] A. Leitsch, The Resolution Calculus, Springer Verlag, 1997.

[6] J.W. Lloyd, Foundations of Logic Programming, Springer Verlag, 1987.

[7] D.W. Loveland, Automated Theorem Proving: A logical basis, North Holland, 1978.

[8] D. Mackenzie, The Automation of Proof: A Historical and Sociological Exploration, Annals of the history of computing, **17**, 3, 1995.

[9] J. Malitz, Introduction to Mathematical Logic, Springer Verlag, 1984.

[10] W. McCune, OTTER 3.0 Reference Manual and Guide, Technical Report, UC-405, ANL-94/6, Argonne National Laboratory, 1995.

[11] B. Meltzer, The Use of Symbolic Logic in Proving Mathematical Theorems by means of a Digital Computer, 1969.

[12] D. Prawitz, Advances and Problems in Mechanical Proof Procedures, Machine Intelligence **4**, 59-71, 1969.

[13] J.A. Ramírez, Un Acercamiento a la Demostración Automática de Teoremas, Tesis de licenciatura, UNAM, 1996.

[14] J.A. Robinson, A Machine Oriented Logic Based on the Resolution Principle, Journal of the ACM **12**, 1, 23-41, 1965.

[15] S. Sabre, Traducción Automática a Forma Clausular, Tesis UNAM, 1988.

[16] S. Sabre, C. Loyo, Mejoras al Algoritmo Clásico de Skolemización, Aportaciones Matemáticas **6**, SMM, México, 1989.

[17] R. Shostak, An Algorithm for Reasoning About Equality, Communication of the ACM, Vol.**21**, No.7, 1978.

[18] V. Sperschneider, G. Antoniou, Logic a Foundation for Computer Science, Addison-Wesley, 1991.

[19] H. Wang, Computer Theorem Proving and Artificial Intelligence, Contemporary Mathematics **29**, 1984.

[20] L. Wos, The Automation of Reasoning an Experimenter's Notebook with OTTER Tutorial, Academic Press, 1996.

[21] L. Wos, R. Overbeek, E. Lusk, J. Boyle, Automated Reasoning Introduction and Applications, McGraw-Hill, 1992.

[22] L. Wos, G. Robinson, Carson, D., Efficiency and Completeness of the Set of Support Strategy in Theorem Proving, J.A.C.M., **12**, 1965.