UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

UNA PRIMERA APROXIMACION A LA DEMOSTRACION AUTOMATICA DE TEOREMAS

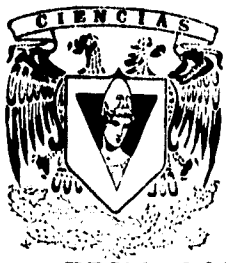
T E S I S

QUE PARA OBTENER EL TITULO DE :

M A T E M A T I C O

P R E S E N T A :

JOSE ANTONIO RAMIREZ MOGUEL



ASESOR: M. en C. JOSE ALFREDO AMOR MONTAÑO

TESIS CON
FALLA DE ORIGEN

UNIVERSIDAD NACIONAL
AVENOMA DE
MEXICO

M. EN C. VIRGINIA ABRIN BATULE
Jefe de la División de Estudios Profesionales
Facultad de Ciencias
P r e s e n t e

Los abajo firmantes, comunicamos a Usted, que habiendo revisado el trabajo de Tesis que realiz(ó)ron el pasante(s) Ramírez Moguel José Antonio

con número de cuenta 8657177-7 con el Título:

"Una primera aproximación a la Demostración Automática de Teoremas"

Otorgamos nuestro **Voto Aprobatorio y consideramos que a la brevedad deberá presentar su** Examen Profesional para obtener el título de Matemático

GRADO	NOMBRE(S)	APELLIDOS COMPLETOS	FIRMA
M. en C.	José Alfredo	Amor Montaño	
Director de Tesis			
M. en C.	Miguel	Murguía Romero	
M. en C.	Carlos	Torres Alcaraz	
M. en C.	Juan José	Montellano Ballesteros	
Suplente			
M. en C.	Rafael	Rojas Barbachano	
Suplente			

A mis amigos

Agradezco sinceramente a Malinatzin Netzahualcóyotl
por su valiosa ayuda en la transcripción de este texto,
A Luis Nava por la asesoría en la programación,
y especialmente
a José Alfredo Amor por su gran paciencia conmigo
y enorme interés en este trabajo.

Una primera aproximación a la Demostración Automática de Teoremas

José Antonio Ramírez Moguel
Facultad de Ciencias, UNAM

INDICE

INTRODUCCION

El propósito de la presente tesis es el de dar, de manera accesible, una visión general de lo que se ha llamado Demostración Automática de Teoremas (DAT). Ésta es un área que tiene tanto a la Lógica Matemática como a la Computación como las ciencias que le dan fundamento. Es además un excelente ejemplo propio para evidenciar la gran interrelación que existe entre estas dos ciencias y la forma en que pueden interactuar arrojando resultados por demás interesantes para ambas partes.

La presente tesis se dedicará pues, a establecer un semblante de lo que es la Demostración Automática de Teoremas lo suficientemente riguroso para dar un seguimiento a los resultados que aquí se presentan, pero de manera que el lector sea llevado directamente a los temas necesarios para un entendimiento de esta área. Así pues, se omitirán algunas demostraciones que pueden ser revisadas en la mayoría de los libros de texto básicos de Lógica Matemática y se enfocará la tesis más a atacar el problema desde un punto de vista práctico.

Asimismo, se incluye una propuesta de un algoritmo para la ejecución semi-automática de desmostraciones dentro de la Lógica Matemática y su implementación en lenguaje C. Dicha propuesta tratará de ser "eficiente" en relación al problema que se desea atacar, es decir, la demostración de un teorema. La eficiencia de la que se habla, es la razón por la que no se propone un demostrador totalmente automático. Esto se podrá entender con mayor facilidad en relación a las observaciones que se harán en los capítulos I y II de la presente y son consideraciones de caracter totalmente práctico.

CAPITULO 1

Conceptos y resultados preliminares de Lógica Matemática

Antecedentes

El lenguaje de la Lógica Matemática está constituido por fórmulas llamadas Fórmulas Bien Formadas. Estas son las "frases" de la Lógica y, como todo lenguaje, se puede estudiar conforme a dos ramas de la Lingüística diferentes:
La primera es la Semática, que estudia a las fórmulas o frases por el significado que tienen, y los resultados de este estudio se encuentran en la Teoría de Modelos. La otra forma es la Sintáctica que se dedica a estudiar la estructura gramatical o funcional del Lenguaje. El método sintáctico está más bien relacionado con el uso de gramáticas o sistemas formales (que son estructuras matemáticas). Estos últimos se pueden ver como mecanismos de producción del lenguaje. Son de especial importancia aquellos sitemas formales que a partir de un conjunto de fórmulas llamadas axiomas generan a todas aquellas fórmulas que son universalmente válidas (U.V:)(desde el punto de vista semático). Así, mediante "reglas bien definidas" se pueden obtenter "resultados ciertos". A esta área de la Lógica Matemática se le llama Demostración Automática de Teoremas (DAT).

Definición del Lenguaje de Primer Orden.

A continuación se da una definición del Lenguaje de Primer Orden de la Lógica Matemática. Como cualquier lenguaje éste se encuentra constituido por símbolos:

Definición: El alfabeto del lenguaje de la lógica de primer orden consta de dos tipos de símbolos. Los llamados **Símbolos lógicos** que son:

1. Variables: $x_1, x_2, x_3, \ldots$

2. Conectivos: $\vee$, $\wedge$

3. Cuantificadores: $\exists$, $\forall$

4. Negación: $\neg$

5. Símbolo de igualdad: $\approx$

6. Símbolos de puntuación: $(,),,$

Además, un conjunto de símbolos no lógicos que tienen la capacidad de representar constantes, funciones y relaciones entre los elementos a los que hace referencia una fórmula, esto último desde el enfoque semático. Desde el sintáctico sólo son otros símbolos del lenguaje.

7. Símbolos de funciones: $f_1^{n_1}$, $f_2^{n_2}$, $f_3^{n_3}$, ...

8. Símbolos de predicado: $P_1^{n_1}$, $P_2^{n_2}$, $P_3^{n_3}$

9. Símbolos de constantes: c_1, c_2, c_3, ...

En una fórmula se emplean tantos de estos símbolos como sean requeridos para expresar completamente el contenido semático deseado para ésta. Semáticamente las variables y constantes permiten representar a diversos elementos de un conjunto llamado **Universo de Interpretación.** Como su nombre lo indica, las variables podrán tomar diferentes valores de representación mientras que las constantes representan a elementos distinguidos de este conjunto. Las variables pueden ser modificadas por cuantificadores como son ∃ (existencial) y ∀ (universal).
Además de las variables y constantes los símbolos funcionales nos dan también la facultad de referirnos indirectamente a elementos del Universo de Interpretación. Estos símbolos, como su nombre lo dice, representan funciones que toman valores en los elementos del Universo y son formas de referencia indirecta a éstos. A las cadenas de símbolos del lenguaje que representan elementos del Universo de Interpretación se les llama **términos** y se definen inductivamente.

Definición. Un **término** es:
- Una variable x_i o una constante c_i, para alguna i N

- Una cadena de la forma $f_i^{n_i}(t_1,...,t_{n_i})$, en donde $t_1,...,t_{n_i}$ son términos.

Una cadena de símbolos del lenguaje es un término sólo si lo es en base a los incisos anteriores.

Los símbolos de predicado sirven para representar propiedades que pueden tener los elementos del universo o ciertas relaciones entre ellos. Una de estas relaciones, que posee un gran significado especial, definido de antemano, es la de la igualdad, entendida como identidad, y es representada por el símbolo "≈".

Los conectivos "∧", "∨" y la negación "¬" sirven para crear, a partir de fórmulas, otra fórmula más compleja. Representan lo que entendemos por conjunción (y) "∧" ó disyunción (o) "∨". La negación "¬" representa lo mismo que la palabra "no" en lenguaje natural, y hace que lo que se encuentre después de ella tome el significado contrario (en sentido semático) a la fórmula original. Con estos símbolos y con los términos se definen las llamadas **Fórmulas Bien Formadas,** que constituyen los elementos o Frases del lenguaje de Pimer Orden.

Las Fórmulas Bien Formadas son las cadenas de estos símbolos que constituyen el Lenguaje de Primer Orden. Las fórmulas se definen al igual que los términos, inductivamente:

Definición. Una Fórmula Bien Formada es:

- Una cadena del tipo $P_i^{n_i}(t_1,\ldots,t_{n_i})$, o del tipo $t_i \approx t_j$

En donde $t_1,\ldots,t_{n_i},t_i,t_j$ son términos

Estas fórmulas son llamadas Fórmulas Atómicas.

- Las cadenas de la forma:

$$(F_i \wedge F_j)\ (F_i \vee F_j)\ (\neg F_i)\ (\exists x_i F)\ (\forall x_i F)$$

en donde F,F_i,F_j son fórmulas.

Una cadena de símbolos es una Fórmula Bien Formada sólo si lo es en base a los incisos anteriores.

Las fórmulas bien formadas de la forma $(\neg F_i \vee F_j)$ se abrevian como $(F_i \rightarrow F_j)$

Las fórmulas del tipo $(\neg F_i \vee F_j) \wedge (\neg F_j \vee F_i)$ se abrevia $(F_i \leftrightarrow F_j)$

De esta manera quedan definidos enequívocamente los elementos del lenguaje correspondiente a cierto conjunto de propiedades y funciones en particular. Estas se llaman Fórmulas o Fórmulas Bien Formadas.

Por ejemplo, la cadena de símbolos siguientes constituye una fórmula:

$$\left(\left(\left(\forall x_1\left(\neg\left(\exists x_2 P_1^2(x_1,x_2)\right)\right)\right) \vee P_2^0\right) \vee \left(\neg P_1^2\left(f_1^1(x_3), f(x_3)\right)\right)\right)$$

La siguiente no es una fórmula:

$$\wedge \exists x_1 x_2, \forall))$$

Como una convención para simplificar el lenguaje se pueden utilizar las letras **P, Q, R, S, T,** etc. para símbolos de predicado y **f, g, h, l, m, n,** etc., para símbolos de función. Además se pueden omitir algunos paréntesis, siempre y cuando no se haga ambigua su lectura.

Como ya se mencionó, dado un lenguaje de este tipo, es posible asignar a él un conjunto de elementos o Universo de Interpretación, una interpretación para los símbolos no lógicos y una interpretación canónica de los símbolos lógicos. Todo esto se denomina una **interpretación del lenguaje** y de esta manera es posible dar a cada fórmula bien formada de este lenguaje un significado preciso y un valor de verdad para esta interpretación, (es decir, verdadero o falso). Haremos notar que el valor de verdad de una fórmula depende de la interpretación, (es decir, verdadero o falso). Haremos notar que el valor de verdad de una fórmula depende de la interpretación dada para ésta.

Definición. A una interpretación en donde una fórmula **F** es verdadera se le llama un **Modelo de F.**

Definición. Se llama **Universalmente Válida U.V.** a una fórmula F si es verdadera en cualquier interpretación posible del lenguaje del tipo que emplea la fórmula.
La validez universal de una fórmula F se denota:

$$\models F$$

Definición. Una fórmula F se llama insatisfacible si ninguna intepretación de su lenguaje la hace verdadera.

Teorema. Una fórmula F es Universalmente Válida si, y sólo si, su negación $\neg F$ es insatisfacible.

El objetivo de la presente tesis, como ya se mencionó, es el de dar un procedimiento que identifique fórmulas universalmente válidas. El teorema anterior expone las bases de otro método para descubrir si una fórmula es universalmente válida, ya que, es equivalente a que la **negación** de esta fórmula sea **insatisfacible**, entonces, bastará con analizar esta negación. A los procedimientos de demostración automática que utilizan la negación de fórmulas les llamaremos **procedimientos de refutación**.

El problema que ataca esta tesis es, pues, el de dar un procedimiento para descubir si una fórmula φ es U.V. o, lo que es lo mismo, descubrir si $\neg\varphi$ es insatisfacible.

Desde el enfoque sintáctico del lenguaje se definen también ciertos conceptos, ya no para darle un significado a las fórmulas, sino pra formalizar su gramática. Para esto se usan las Gramáticas o Sistemas Formales.
Un **Sistema Formal** es un mecanismo que define de manera simple las reglas sintácticas que están permitidas dentro de un lenguaje determinado (o sea, las reglas que generan al lenguaje). Los definimos como un mecanismo en el sentido de que , por su forma, nos permite realizar cambios en las cadena del lenguaje de manera sistemática sin necesidad de una comprensión del significado de su acción.

Un Sistema Formal se define como:

1. Un conjunto decidible de fórmulas bien formadas del lenguaje llamadas **axiomas**.
2. Un conjnto de reglas claras no ambiguas que nos llevan de modo **decidible** de algunos elementos del lenguaje en otros. Estas son nombradas **Reglas de inferencia**.

Se llama un **Teorema** al resultado de aplicar un número finito de veces, reglas de inferencia a axiomas o a fórmulas demostradas con anterioridad, es decir otros teoremas previamente demostrados.
Al conjunto de fórmulas que constituyen el desarrollo del teorema es un sistema formal determinado se le llama **Demostración** de éste.

La definición de un teorema, dado un sistema formal, se puede dar, entonces, inductivamente de la siguiente forma:

Definición. Los teoremas de un sistema formal **S** son:
1. El conjunto de axiomas
2.El resultado de aplicar una regla de inferencia a axiomas o a otros teoremas.

Más que esto, el concepto que se formaliza con la definición de sistema formal, es el de proceso mecánico o sistemático, como quiera que se entienda esto. La tesis que se maneja aquí es la de que todo proceso mecánico puede teorizarse mediante un sistema formal y viceversa: todo sistema formal corresponde a un proceso mecánico. Por lo menos esto último resulta evidente ante la definición dada.

Esto último nos hace pensar en la posibilidad de que alguna máquina que maneje información, como lo son las computadoras, pueda realizar esta manipulación sintáctica. La liga entre las computadoras y los sistemas formales es muy estrecha. La presente tesis se enfocará a enfatizar este hecho y a buscar formas en que esta manipulación se haga de manera práctica.

A continuación se dará un ejemplo de un sistema formal para la Lógica de Primer Orden o Lógica de Predicados:

1. Un conjunto de axiomas que consta de Fórmulas Bien Formadas del lenguaje, de manera que puedan ser identificadas con precisión:

1.1 El conjunto de tautologías *

1.2 Las fórmulas de la forma: $\left(\forall x\left(\alpha \rightarrow (\alpha)^{x}_{t}\right)\right)$ en donde t es sustituible por x en α

1.3 Las fórmulas de la forma: $\left(\forall x(\alpha \rightarrow \beta) \rightarrow \left((\forall x\alpha) \rightarrow (\forall x\beta)\right)\right)$

1.4 $\left(\forall x(x \approx x)\right)$

1.5 Las fórmulas de la forma: $\left(\forall x,y\left(x \approx y \rightarrow \left(\alpha(x) \rightarrow \alpha\,{}^{x}\!/_{y}\right)\right)\right)$ en donde $\alpha\left({}^{x}\!/_{y}\right)$

denota el resultado de sustituir x por y en algún lugar de α.

1.6 Las fórmulas de la forma: $\left(\alpha \rightarrow (\forall x\alpha)\right)$ en donde x no ocurre libre en α.

* En este inciso se utilizan esquemas de axiomas, en donde cada esquema representa a un número infinito de axiomas. Aún así, no se pierde la decidibilidad del conjunto de axiomas, ya que estos pueden ser fácilmente reconocidos por la simple observación de su forma.

2.- La regla llamada **Modus Ponens** que aplica a un conjunto de dos fórmulas y tiene como resultado otra fórmula. Esta es una regla conocida desde hace mucho tiempo y posee la cualidad de que si las dos formulas a las que se aplica son verdaderas (en sentido semático) la fórmula resultante será también verdadera.
Se esquematiza de la siguiente manera:

$$\frac{F_i \vee \neg F_j \qquad F_j}{F_i}$$

En cuanto a las tautologias, el problema de identificarlos está garantizado por el isomorfismo entre éstas y las fórmulas universalmente válidas del cálculo o lógica proposicional, es decir, aquella en la que sólo se hace referencia a predicados, en donde no intervienen las variables ni las constantes del conjunto universal. Para este tipo de fórmulas existen procedimientos de demostración que resultan ser completos (o sea que pueden demostrar la validez universal de cualquier tautología), y que en caso de no serlo también darán una respuesta negativa. A este tipo de procedimiento se le denominará **algoritmo**. Pueden consultarse diversos algoritmos para la validez universal del cálculo proposional, como el de Tablas de Verdad, Arboles de Verdad y el algoritmo para la identificación de tautologías: el algoritmo Davis-Putnam para el cálculo proposicional.

La existencia de reglas de inferencia que preserven la veracidad en sentido semático nos hace pensar en una relación entre los dos conceptos de los que hemos hablado: verdad vs. mecanismo formal. Esta relación existe y fue demostrada por el famoso lógico Kurt Gödel en la década de los treinta:

Teorema de Completud y Correctez de la Lógica de Primer Orden.
Los teoremas del sistema formal anterior se corresponden exactamente con las fórmulas universalmente válidas del lenguaje.

Este último metateorema constituye el pila de la Demostración Automática, ya que sería imposible hacer este tipo de manejo de fórmulas si no hubiera una relación entre la verdad de las fórmulas y los procesos de manejo simbólico de éstas (demostraciones).

Formas Normales

A continuación, se dan un conjunto de transformaciones para fórmulas bien formadas que nos permiten realizar una manipulación simbólica de las fórmulas de una manera más simple. Estas transformaciones están justificadas por una serie de metateoremas cuya demostración se puede encontrar en [Malitz].

A lo que se pretende llegar, dada una fórmula es a una fórmula **lógicamente** equivalente cuyas literales no dependen una de otra. en cuanto a **cuantificadores** o conectivos. Además se encuentra ordenada de manera que facilite su manipulaciòn simbólica. A esta forma la llamaremos **forma clausular.**

Las siguientes transformaciones van del conjunto de Fórmulas Bien Formadas a él mismo y preservan siempre la satisfacibilidad.

La primera de estas transformaciones preparatorias de una fórmula **es la Forma Normal de Skolem para Satisfacción** de una fórmula. Dada una fòrmula , **existe** otra fórmula denotada **FNSS** ( ) que tiene en el lugar de cada variable **cuantificada** existencialmente una nueva constante o una nueva función, adicional **al lenguaje**, que depende exclusivamente de las variables libres o cuantificadas universalmente.

Ejemplo:

Sea φ

$$\exists x \exists y \left(\left(P(x) \vee P(y) \right) \wedge \forall z \exists w \left(P(w) \wedge P(x) \wedge \neg P(z) \right) \right)$$

FNSS(φ):

$$\left(P(a) \vee P(b) \right) \vee \left(\forall x \left(P(Sk1(z)) \wedge P(a) \wedge P(z) \right) \right)$$

Sk1 ( ) es una de las mencionadas funciones de Skolem que sustituye a la variable "w" (cuantificada existencialmente), y en este caso depende de el valor que tome la variable cuantificada universalmente ("z"). La cuantificación existencial de "x" es sustituida por una constante "a" y la de "y" por la constante "b".

El proceso por el cual se llega a obtener la FNSS (φ) se llama Skolemización y es posible realizarla algoritmicamente. Este algoritmo se puede consultar en cualquier libro de texto de Lógica Matemática, por ejemplo [Malitz].

Contando con este algoritmo podemos aseverar que para cada fórmula del lenguaje podemos encontrar otra fórmula que está libre de cuantificaciones existenciales y que, gracias al Teorema de Skolem, sabemos que conserva su satisfacibilidad:

Teorema de Skolem. Una fórmula φ es satisfacible si, y sólo si FNSS (φ) lo es.

Los procedimientos de refutación utilizan esta forma normal de Skolem aplicada a la negación de la fórmula que quiere demostrar automáticamente. Lo que se logra con esta fórmula normal es liberar a una fórmula de sus cuantificadores existenciales sin que ésta pierda su significado *:

φ es U.V. si y sólo si $\neg\varphi$ es insatisfacible

$\neg\varphi$ es insatisfacible si y sólo si FNSS($\neg\varphi$) lo es

Entenderemos por un significado el valor de verdad (verdadero o falso) que toma una fórmula ante las diversas interpretaciones de ella. De esta manera dos fórmulas tendrán el mismo significado si y sólo si, cualquier interpretación que haga verdadera a la primera también lo hará con la otra. Llamaremos a esta relación entre fórmulas equivalencia lógica y se simbolizará:

$$\varphi \equiv \phi$$

(se lee φ es lógicamente equivalente a ϕ)

También se definirá otra relación entre las fórmulas que resulta ser un debilitamiento de la anterior. Llamaremos a dos fórmulas φ y ϕ lógicamente equiposibles si φ es satisfacible cuando, y sólo cuando ϕ también lo es. O, dicho de otra manera:

φ es insatisfacible si, y sólo si ϕ lo es

y se simboliza:

$$\varphi \overset{E}{=} \phi$$

Es claro que ésta es una relación de equivalencia, también que es una relación más debil que la de equivalencia lógica, ya que i dos fórmulas son lógicamente equivalentes, también serán lógicamente equiposibles, pero no a la inversa:

$$\exists x P(x) \overset{E}{=} P(c); \text{ pero } \exists x P(x) \overset{E}{\neq} P(c)$$

Otras transformaciones a fórmulas que serán de utilidad en estos procedimientos, son la Forma Normal Prenex y las formas Normales Conjuntiva y Disyuntiva. La primera de éstas, tiene la cualidad de tener todos los cuantificadores que aparecen al principio de la fórmula. Por ejemplo:

$$\varphi: \left(P(a) \vee \neg P(b)\right) \wedge \left(\forall z\left(P\left(Sk1(z)\right) \wedge P(a) \wedge \neg P(z)\right)\right)$$

$$\text{FNP}(\varphi): \forall z\left(\left(P(a) \vee \neg P(b)\right) \wedge \left(P\left(Sk1(z)\right) \wedge P(a) \wedge \neg P(z)\right)\right)$$

La Forma Normal Prenex es, pues, otra fórmula que posee todos sus cuantificadores antes que cualquier predicado. Esta forma normal posee la cualidad de preservar no sólo la posibilidad sino también el significado, para toda fórmula φ:

Teorema de existencia de la Forma Normal Prenex:
Para toda fórmula φ , existe una Forma Normal Prenex asociada **FNP()**, que se puede obtener por medio de un algoritmo y que cumple:

Todo modelo de. φ es modelo de **FNP(φ)** e inversamente ó:

$$\varphi \equiv \text{FNP}(\varphi)$$

Las siguientes transformaciones que usaremos son las Formas Normales Conjuntiva y Disyuntiva de una fórmula . Denotadas **FNC(φ)** y **FND(φ)** respectivamente.

La **Forma Normal Disyuntiva** de una fórmula φ es una fórmula que tiene la siguiente estructura:

$$\left(\varphi_{11} \wedge \varphi_{12} \wedge \ldots \wedge \varphi_{1n_1}\right) \vee \ldots \vee \left(\varphi_{m1} \wedge \varphi_{m2} \wedge \ldots \wedge \varphi_{mn_m}\right)$$

En donde cada φ_{ij} es una fórmula atómica o la negación de una fórmula atómica. Estas últimas se llamarán de ahora en adelante **literales**. Esta forma normal tiene, al igual que la anterior, la cualidad de ser lógicamente equivalente a su forma original. Entonces, para cualquier fórmula φ:

$$\varphi \equiv \text{FND}(\varphi)$$

La contraparte dual de esta forma es la **Forma Normal Conjuntiva o Forma Clausular** cuya estructura es:

$$\left(\varphi_{11} \vee \varphi_{12} \vee \ldots \vee \varphi_{1n_1}\right) \wedge \ldots \wedge \left(\varphi_{m1} \vee \varphi_{m2} \vee \ldots \vee \varphi_{mn_m}\right)$$

Donde cada φ_{ij} es una literal.

Al igual que la Forma Normal Disyuntiva, ésta es equivalente a la original:

$$\varphi \equiv \text{FNC}(\varphi)$$

Ejemplo:

φ: $\forall z\left(\left(P(a) \wedge Q(b)\right) \vee P(z) \vee \neg R(a) \vee \neg P(b)\right)$

FNC(φ): $\forall z\left(\left(P(a) \vee P(z) \vee \neg R(a) \vee \neg P(b)\right) \wedge \left(Q(b) \vee P(z) \vee \neg R(a) \vee \neg P(b)\right)\right)$

Como se puede observar, el camino que se toma para llegar a esta forma es el de aplicar las leyes de distribución de la conjunción y de la disyunción. A cada uno de los elementos de la conjunción se les llama cláusulas, y, como ya se mencionó, a cada parte de una cláusula se denomina literal (atómica o negación de atómica). Por último se expondrá un teorema que permite eliminar de una fórmula los cuantificadores universales que aparecen en ella sin perder su significado semántico.

Teorema. Una fórmula con cuantificadores universales de la forma $\forall x_1 x_2 \ldots x_n \varphi$ es verdadera en una interpretación si, y sólo si φ es verdadera en esa misma interpretación.

Ejemplo:

φ: $\forall z\left(\left(P(a) \vee \neg P(b)\right) \wedge \left(P(Sk1(z)) \wedge P(a) \wedge \neg P(z)\right)\right)$

FNSCU(φ) $\left(P(a) \vee \neg P(b)\right) \wedge \left(P(Sk1(z)) \wedge P(a) \wedge \neg P(z)\right)$

Los metateoremas y las definiciones hasta ahora dados, justifican cierta simplificación de las fórmulas. En resumen, lo que se tiene hasta el momento es una fórmula φ:

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow T$$

De la cual se intenta averiguar su Validez Universal. Ésto como se vió, es equivalente a mostrar la insatisfacibilidad de ¬φ:

φ es U.V. si y sólo si ¬φ es insatisfacible

La preparación que a continuación se les dá a las fórmulas es:

-Primero se construye su Forma Normal de Skolem para satisfacción, de manera que es insatisfacible si y sólo si FNSS(¬φ)

-La forma Prenex de esta última es lógicamente equivalente a ésta, y, entonces garantizamos que:

FNSS(¬φ) es insatisfacible si y sólo si FNP(FNSS(¬φ)) lo es.

-Por último, se obtiene la Forma Normal Conjuntiva o Clausular y se eliminan los cuantificadores universales (Forma Normal Sin Cuantificadores Universales FNSCU) Así tenemos:

FNSCU (FNC(FNSS(¬φ)))es insatisfacible

si y sólo si FNP (FNSS(¬φ)) lo es.

y por lo tanto, esta última fórmula es insatisfacible si y sólo si φ es una fórmula universalmente válida.

La fórmula que se obtiene al final de esta serie de transformaciones tiene las siguientes propiedades:
- No posee cuantificadores
- Está en forma clausular
- Es equiposible lógicamente a ¬φ y por lo tanto es insatisfacible si y sólo si φ es U.V.

Los siguientes teoremas son fundamentales para el campo de la DAT proporcionan un método para probar la insatisfacibilidad de una fórmula dada.

El Teorema de Herbrand

Antes de enunciar este teorema, es preciso dar algunas definiciones.

Definición. Una instacia de sustitución de una cláusula o una cláusula instanciada es el resultado de sustituir las variables de dicha cláusula por términos constantes. Estos últimos se definen inductivamente como sigue, es decir, un **término constante** es:
- una constante ó
- una función aplicada a términos constantes.

Definición. Un enunciado sin cuantificadores es una fórmula sin cuantificadores, que sólo contiene términos constantes.

El siguiente teorema, el de Herbrand, es considerado por muchos, la piedra angular de la demostración automática. Este teorema provee un método para la comprobación de la insatisfacibilidad de una fórmula del lenguaje. En éste se reduce la necesidad de probar un número infinito de interpretaciones posibles de la fórmula a una simple revisión de satisfacibilidad de un número finito de fórmulas simples.
Aquí tendremos que recordar que el tratamiento mencionado en las secciones anteriores transforma cualquier fórmula del lenguaje de primer orden en una conjunción de cláusulas, es decir, en una conjunción de disyunciones de literales (fórmulas atómicas o negaciones de fórmulas atómicas). Esta conjunción de cláusulas no tiene cuantificadores y puede o no tener variables. Una instancia de sustitución de una conjunción de cláusulas es claramente un enunciado sin cuantificadores.

Teorema de Herbrand. Una fórmula en forma clausular es insatisfacible si, y sólo si hay un conjunto finito de cláusulas instaciadas inconsistentes entre sí. (contienen una fórmula inconsistente)

Ejemplo, Sea φ:

$$\exists x \exists y \left(\left(P(x) \vee P(y) \wedge \forall z \exists w \left(P(w) \wedge P(x) \wedge \neg P(z) \right) \right) \right)$$

Será convertida en:

$$\left(P(a) \vee \neg P(b) \right) \wedge P\left(Sk1(z) \right) \wedge P(a) \wedge \neg P(z)$$

o mejor:

$P(a) \vee \neg P(b)$	cláusula 1
$P\left(Sk1(z) \right)$	cláusula 2
$P(a)$	cláusula 3
$\neg P(z)$	cláusula 4

Esta fórmula es insatisfacible ya que las siguientes instancias así lo indican.
Obsérvese que sólo las cláusulas 2 y 4 tienen variables; las cláusulas 1 y 3 ya están instanciadas.

En el ejemplo.

Instancias de sustitución:

¬P(a)	cláusula 4 (z/a)
P(a)	cláusula 3

Esta subfórmula es inconsistente.

Este teorema fue demostrado originalmente de manera sintáctica en 1930 y esa demostración es de gran complejidad, sin embargo, los libros modernos utilizan para explicarlo, la claridad de la herramienta semática y la relación entre estas dos, dada por el Teorema de Completud de Gödel. su demostración puede ser revisada en cualquier libro de lógica. P. ej. [Malitz].

El otro teorema que termina la justificación de que este método de demostración automática algoritmico, es decir, siempre da una respuesta correcta (la fórmula es satisfacible o no) en un tiempo finito. Dicho de otra manera: es un problema **decidible**.

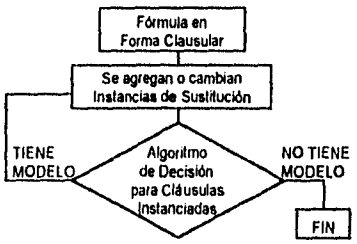
Teorema. La satisfacibilidad de un enunciado sin cuantificadores es decidible. [Loveland].

Lo anterior significa que hay un algoritmo o proceso efectivo de decisión tal, que dado cualquier enunciado sin cuantificadores, decide si éste es satisfacible o no lo es, de modo efectivo y en tiempo finito. Con este último teorema se cierran los antecedentes que justifican a la DAT.

El método implícito en el teorema de Herbrand junto con este último será entonces:

1. Ralizar instancias de sustitución de las cláusulas de la fórmula utilizando las constantes y funciones que aparecen en ella.
2. Checar si el enunciado sin cuantificadores resultante de la conjunción de estas cláusulas es satisfacible o no lo es.
3.1 Si es satisfacible se agregan o cambian* nuevas isntancias al conjunto que ya se tiene y se ejecuta el paso 2.
3.2 En caso de ser instisfacible, entonces el conunto de cláusulas también lo es (por el teorema de Herbrand)

Representaremos el procedimiento con el siguiente diagrama:



* Al añadir instancias de cláusulas al conjunto de cláusulas instanciadas, no se modifican las hipótesis del teorema de Herbrand ni su aplicación, ya que si B es insatisfacible (B es un conjunto de cláusulas instanciadas) también lo es B C para cualquier C, conjunto de cláusulas instanciadas. Por lo tanto, es conveniente agregar nuevas instancias al conjunto, dejando las que ya estaban, pues así, es posible obtener la contradicción buscada.

CAPITULO 2

Historia de la demostración automática desde 1954

Antes de comenzar con una breve revisión del campo de la demostración automática de teoremas (DAT) dentro de la Lógica Matemática o de las Ciencias de la Computación sería conveniente poner en claro algunos conceptos que se utilizarán en el transcurso del capítulo.
Empezaremos por dar una definición de lo que se entenderá por Demostración Automática de Teoremas: según Loveland [Loveland], la demostración automática de teoremas es el uso de computadoras como ayuda para efectuar cálculos no numéricos, es decir, aquellos sobre la validez de expresiones de algún lenguaje simbólico.

Si conservamos la anterior definición, procedemos entonces a revisar algunos modelos de Demostración Automática. La evolución de estos modelos puede ser considerada como natural, con base en el estado reinante de la teoría de la Lógica Matemática y de la Computación, en los años cincuenta, fecha en que tiene inicio.

Deducción Formal

El demostrador que parece ser el más "natural" es la deducción formal ejecutada por una máquina. Si observamos la definición de un sistema formal, encontraremos que en base a un conjunto conocido de axiomas y, mediante la aplicación de ciertas reglas de inferencia, se puede llegar a demostrar cualquier teorema. Ya con esto, podemos concebir naturalmente a un autómata que siga los pasos antes mencionados, convirtiéndose así, en un generador de teoremas, una máquina que produce una lista de todos los teoremas de una teoría determinada por los axiomas y las reglas de inferencia con los que se alimente. Sin embargo, esto no da un proceso efectivo para generar un teorema determinado de antemano.

Metodos de Herbrand

Otros procedimientos basados también en la Lógica Matemática son aquellos que utilizan los resultados del Teorema de Herbrand, como ya se mencionó garantiza la demostración (o en su forma dual, la insatisfacibilidad) de una fórmula mediante la revisión de un conjunto de cláusulas instanciadas. Mediante la sustitución de sus variables por un número finito de elementos pertenecientes a los llamados Univesos de Herbrand. Los **Universos de Herbrand** están constituidos pro todos los términos constantes posibles en el lenguaje de la fórmula. Este método, como se verá, es probablemente el de mayor importancia en el desarrollo de DAT.

Uso de heurísticas

Los dos métodos anteriores son formas de manipulación simbólica. la deducción formal y los métodos de Herbrand toman en cuenta para su ejecución sólamente los símbolos que aparecen en la fórmula sin siquiera darles una interpretación o significado. Sin embargo, la forma en que un matemático demuestra un teorema no

esta excenta de interpretación de simbolos que aparecen en la fórmula; asi como tampoco del uso de la experiencia ni de la intuición. Es por esto que las heurísticas constituyen una herramienta más para la demostración automática. Entenderemos por heuristicas a las reglas o ayudas de los tipos mencionados anteriormente. Dentro de estas heuristicas podemos mencionar la intervención del matemático como apoyo a un demostrador de teoremas, lo que es llamado en inglés "Man-Machine Aproach", es decir, la intereacción del hombre con una computadora para la solución de problemas planteados.
Hasta ahora se han mencionado tres métodos de demostración automática, pero, para obtener algoritmos de demostración automática que sean prácticos se utiliza siempre la combinación de métodos. Es una de las medidas usadas frecuentemente al construir un demostrador que sea efectivo y eficiente* . Como se verá más adelante, desde los inicios de la DAT no se ha dado una separación clara entre estos métodos; ni se ha dado una preferencia por alguno.

1954-1965

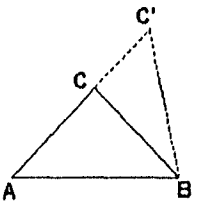
En esta seccion se describirán someramente los avances en "Demostración Automática de Teoremas" (DAT) desde sus primeras aplicaciones hasta 1965, una fecha muy significativa par alos interesados en el área, ya que denota la aparición de la regla Resolución de J.A. Robinson. Esta regla tiene y ha tenido gran influencia en el campo y ha sido utilizada por muchos investigadores quienes han inventado gran cantidad de refinamientos con diversas apliacaciones.

A medidados de los años cincuenta, el desarrollo de las computadoras permitió a las universidades el acceso a las primeras máquinas. Estas fueron rápidamente solicitadas por científicos que, con diversos proyectos comenzaron a obtener beneficios de su potencialidad. En Lógica Matemática, dado su grado de desarrollo, existía un medio apropiado para desarrollar los primeros demostradores automáticos, pioneros de la Inteligencia Artificial. Surge la idea de utilizar estas poderosas máquinas como instrumentos capaces de realizar el trabajo de demostrar teoremas. Esta idea, acarreada desde siglos atrás por pensadores como Leibniz, quien propueso el desarrollo de un "Calculus Ratiocinator": un cálculo mediante el cual demostrar la verdad o falsedad de la afirmación de cualquier tema, se redujera a una simple operación de cálculo. Esto sólo se hace realizable hasta el surgimiento de máquinas capaces de ejecutar los algoritmos sugeridos, y con la calidad de las computadoras digitales.
Ante esta situación, las ideas desarrolladas en torno al proyecto de Hilbert para la creación de un lenguaje y un cálculo que formalizará a la Lógica, así como los resultados obenidos a partir de ellas, se hicieron inevitablemente atractivas tanto a lógicos como a científicos de la computación.

En 1954, dentro del "Summer Institute for Symbolic Logic" en Cornell University, un congreso de verano en donde se reunieron por primera vez un número considerable de lógicos junto con otro tanto de científicos de la computación, para discutir temas de interés comun, surgen algunas propuestas de desarrollo para la demostración automática. Abraham Robinson, de los laboratorios Argonne en Estados Unidos, propone por primera vez a esta comunidad el utilizar los universos de Herbrand y

algunos teoremas de la Lógica para atacar el problema con el uso de máquinas ordenadoras [Robinson A.]. El mismo Robinson hace algunas sugerencia para la eficiencia de los demostradores que se puedieran construir con ella. La demostración por medio del Universo de Herbrand, como ya se ha visto, involucra la sustitución en la fórmula de un cierto número de elementos de este universo o modelo hasta encontrar una refutación. Algunas de estas sustituciones son innecesarias para la demostración. Es más, las sustituciones relevantes tienen cierta relación con lo que llamamos intuición matemática; es decir, de alguna manera observamos al hacer una demostración ciertos elementos de un modelo que nos permite llegar a una demostración. Robinson ejemplifica esta situación con un teorema de la Geometría:

Sea ABC un triángulo cualquiera:



Sea C' tal que BC=BC'

Se desea demostrar el siguiente teorema:

Teorema:

$$AB + BC \geq AC$$

La suma de los lados AB y BC es mayor o igual a AC

Demostración:

$$\angle ACC' \geq \angle AC'C$$

$$\therefore AC' \geq AC$$

$$\text{Pero } AC' = AB + BC' = AB + BC$$

$$\therefore AB + BC \geq AC$$

Teorema Auxiliar: En un triángulo cualquiera, si $\angle 1$ y $\angle 2$ son ángulos y $\angle 1 \geq \angle 2$, entonces $L1 \geq L2$; donde $L1 \geq L2$ son los lados opuestos a los ángulos $\angle 1$ y $\angle 2$ respectivamente.

También en 1954, Martin Davis utilizando la computadora del Instituto de Estudios Avanzados (IASC o Johniac, de cariño, en honor de John von Neumann) implementó por primera vez un algoritmo para la demostración automática. El procedimiento

utilizado es el algoritmo de Presburger para la aritmética aditiva: Este algoritmo garantiza el cálculo del valor de verdad de proposiciones en las que se utilizan igualdades y congruencias módulo algún entero mediante la transformación de toda proposición a otra equivalente, pero que no contiene cuantificadores. Los resultados de este primer acercamiento computacional fueron pobres. Fue un gran logro que la máquina demostrara que la suma de dos pares es par. Sin embargo, la computadora de la que estamos hablando tenia una capacidad de 1024 palabras de 40 bits, lo que equivaldría a unos 5 Kbytes, una capacidad muy pobre comprarada con la de las actuales computadoras. A pesar de estos difíciles inicios de la DAT, la experiencia acumulada por científicos y programas como estos que mencionamos es que se logró llegar a resultados importantes.

Este primer demostrador del que se tiene registro, no utilizó el Universo de Herbrand como base del algoritmo; ni tampoco se hace uso de otros teoremas clásicos de la lógica como el Teorema de Skolem, ni de la forma normal conjuntiva o clausular, técnicas que con el paso de los años se convirtieron más o menos en comunes para cualquier demostrador, pero hay que hacer notar que estamos hablando de los primeros intentos. Algunos otros demostadores usaron el Universo de Herbrand sin una guía para las instancias de sustitución. Además muchos de estos primeros demostradores no utilizaron el Teorema de Skolem que evita el trato con cuantificadores existenciales. Los resultados de estos primeros demostradores sólo probaron teoremas muy simples debido a la explosión combinatoria que se presenta al generar a los Universos de Herbrand.

Tenemos otro ejemplo, el primer artículo que registra a una máquina que demuestra (Davis no registró sus resultados por escrito en 1954). La llamada "*The Logic Theory Machine*", data de 1957. Fue realizada por Newell, Shaw y Simon . Este programa corrió por primera vez en la navidad de 1955 y llegó a demostrar cerca de 40 teoremas de Principia Matemática de Rusell y Whitehead. Su forma de trabajar consistia en una parte algorítmica-lógica y una parte heurística. La máquina trataba de demostrar el teorema haciendo sustituciones en la fórmula, pero si no hallaba una demostración pronto, entonces trataba de atacar un subproblema que pudiera dar luz sobre la solución del teorema. Esta es una idea muy importante que se desarrolla después y que se aplica en otros métodos (se conoce como "backward chaining" o "backtracking"); y que está presente en técnicas de demostración automática como Resolución. Otros conceptos fundamentales que se esbozan por primera vez en este programa son la representación de problemas y el "Matching" (unificación).

En 1958, Gilmore, inspirado por la técnica de E.W.Beth: Tablas Semáticas, se dedica durante algunos años a la realización computacional de un modelo de demostrador. Este método no utilizaba siquiera el Teorema de Skolem, aunque si, el Universo de Herbrand. El algoritmo de demostración que utilizó implicaba el manejo de formas normales disyuntivas, cuyo crecimiento es exponencial al sustituir variables en una fórmula. Los resultados obtenidos por Gilmore fueron moderados, aunque, dada su situación histórica, facilitaron el trabajo para investigadores posteriores. Logra dar demostraciones de fórmulas válidad moderádamente complejas, del cálculo de primer orden, sin igualdad:

$$\exists x \forall y \forall z \Big(\big(((Fy \rightarrow Gy) \leftrightarrow Fx) \wedge \big(((Fy \rightarrow Hy) \leftrightarrow Gx) \wedge \big(((Fy \rightarrow Gy) \rightarrow Hx) \big) \rightarrow (Fz \wedge Gz \wedge Hz) \big) \big) \Big)$$

demostrada en 01 min.

$$\exists x \forall y \forall z \Big(\big(\big(Fyz \rightarrow (Gy \rightarrow Hx) \big) \rightarrow Fxx \big) \wedge \big((Fzx \rightarrow Gx) \rightarrow Hz \wedge Fxy \big) \rightarrow Fzz \Big)$$

demostrada en 1.42 min.
Sin embargo, la siguiente fórmula:

$$\exists x \exists y \forall z \big((Fxy \rightarrow (Fyz \wedge Fzz)) \wedge ((Fxy \wedge Gxy) \rightarrow (Gxz \wedge Gzz)) \big)$$

La cual parece ser tan sencilla como las anteriores, no pudo ser demostrada, a pesar de que se utilizó el mismo método en la computadora durante 21 minutos.

Hao Wang, trabajando en una IBM 704 Data Processing Machine, en los laboratorios IBM en 1958, y posteriormente en los laboratorios Bell (1959-60), desarrolló 3 programas demostradores de teoremas: El primero, enfocado en el cálculo proposicional. Este, probó 220 de los teoremas del principio Matemática, de manera totalmente automática en 37 min. (lo que actualmente serían segundos). Posteriormente, realizó uno para una parte decidible del cálculo de predicados y, después, un tercero que tenía como dominio el cálculo de predicados completo (Lógica de 1er orden).
Los programas utilizaron las técnicas de Herbrand y de Gentzen para mostrar la validez de fórmulas, pero el problema de la explosión exponencial de términos de Herbrand quedaba sin una solución o propuesta de solución viable. Wang sugirió y comenzó a trabajar sobre la línea de reconocimiento de patrones dentro de las sustituciones de variables para evitar este crecimiento del tiempo de ejecución del algoritmo.

Dentro de esta misma línea, en 1960, Martin Davis y Hilary Putman proponen un nuevo algoritmo para manejar las fórmulas instanciadas. Estas últimas se toman en su forma normal conjuntiva o forma clausular, la cual sería adoprada posteriormente por muchos algoritmos. El algoritmo de Davis y Putnam logra reducir el tiempo de búsqueda de una demostración o refutación; mediante la separación de un conjunto de cláusulas en una conjunción de varios conjuntos de estas más simples. Las conclusiones a las que llegan estos investigadores: es necesario proponer una búsqueda en las instancias de sustitución del Universo de Herbrand y es necesario excluir líneas o partes de fórmulas que no sean relevantes a la demostración. Este algoritmo se discutirá posteriormente con mayor detenimiento, siendo en buena medida, parte fundamental del ejemplo construido en la presente tesis.
La solución a los problemas planteados anteriormente por Davis y Putnam serán considerados en 1960 por Dag Prawitz que propone un método para encontrar refutaciones de fórmulas con el uso de igualdades. El avance anterior fue reconocido inmediatamente por Davis quien lo adoptó en un nuevo algoritmo que combina la búsqueda de sustituciones con el algoritmo de [Davis-Putnam].
Otro ejemplo de esta década de desarrollo de la DAT, es el de aquellos investigadores que prefirieron seguir el camino de la demostración automática con heurísticas o con una relación más directa con los modelos de las fórmulas que están demostrando. Un ejemplo de esto lo tenemos en la "Geometry Theorem Proving Machine", desarrollada en 1959 en la que hace notar la gran ayuda que puede ser para el demostador, trazar primero un modelo de los axiomas del teorema que se quiere atacar. Esto último como una guía para encontrar las sustituciones requeridas en la demostración. Los autores demostrador geométrico utilizaron diagramas de problemas geométricos como ayuda del demostrador. Junto con ésto, se usó también el principio del "*backward chaining*"

que consiste en reducir el problema a submetas que, de ser demostradas llevan a la demostración de la proposición.

Al final de estos primeros diez años de demostración automática, en 1964-65, J. A. Robinson enuncia su regla de Resolución, donde se formalizará el concepto de búsqueda de sustituciones con una regla particular llamada unificación matching. Resolución es una derivación de una regla enunciada por Gentzen.

$$\frac{A_1 \vee \ldots \vee A_n \vee L \qquad B_1 \vee \ldots \vee B_m \vee \neg L}{A_1 \vee \ldots \vee A_n \vee B_1 \vee \ldots \vee B_m}$$

La cual a su vez es una generalización de la regla de encadenamiento de los sitemas formales:

$$\frac{\neg A \vee B \qquad \neg B \vee C}{\neg A \vee C} \qquad\qquad \frac{A \rightarrow B \qquad B \rightarrow C}{A \rightarrow C}$$

Como conclusión de estos primeros diez años de demostración automática, podríamos subrayar el hecho de que si bien no se dieron grandes resultados en cuanto la demostración automática, es indudable que gracias a estos primeros experimentos surgieron las ideas de los grandes métodos de demostración automática, que, posteriormente se desarrollaron y que se encuentran aún en desarrollo.

Dos ejemplos

Analizaremos dos procedimientos de refutación de particular importancia. El primero de ellos es el procedimiento Davis-Putnam, uno de los primeros. El segundo es el procedimiento de Resolución que, como se verá posteriormente, da pie a gran parte del trabajo desarrollado en la DAT. Ambos procedimientos se enuncian a los sitemas formales que, como hemos visto, representan a los procesos mecánicos. Los dos permiten la simplificación de un conjunto de cláusulas instanciadas mediante el empleo de reglas que garantizan la equiposibilidad lógica de éstas con el resultado. Después de cierto tiempo, éstas permiten ver claramente la satisfacibilidad o insatisfacibilidad de cualquier fórmula instanciada.

Procedimiento Davis-Putnam

Este procedimiento puede ser descrito como un sistema formal en donde los axiomas son un conjunto de cláusulas instanciadas como estas:

P(c,g(c))	Q(a)	¬P(c,g(c))	cláusula 1
Q(a)	P(c,g(c))		cláusula 2
P(c,c)	Q(a)	¬Pc,g(c))	cláusula 3
¬P(c,c)	Q(a)	¬Pc(g(c))	cláusula 4
Q(a)	¬P(c,c)	P(c,g(c)	cláusula 5
¬Q(a)			cláusula 6

Y las reglas de inferencia son las siguientes:

R1. Se eliminan todas las cláusulas que contengan una tautología, es decir, las que contienen a una literal L y a su negación

Ej:

R1

P(c,g(c))	Q(a)	¬P(c,g(c))			
Q(a)	P(c,g(c))		Q(a)	P(c,g(c))	
P(c,c)	Q(a)	¬Pc,g(c))	P(c,c)	Q(a)	¬Pc,g(c))
¬P(c,c)	Q(a)	¬Pc(g(c))	¬P(c,c)	Q(a)	¬Pc(g(c))
Q(a)	¬P(c,c)	P(c,g(c)	Q(a)	¬P(c,c)	P(c,g(c)
¬Q(a)			¬Q(a)		

R2. Si se encuentra una cláusula que contenga a otra, es decir, si tienen dos cláusulas. $A \vee B$ y A se elimina la primera.

Ej:

R2

Q(a)	P(c,g(c))		Q(a)	P(c,g(c))	
P(c,c)	Q(a)	¬Pc,g(c))	P(c,c)	Q(a)	¬Pc,g(c))
¬P(c,c)	Q(a)	¬Pc(g(c))	¬P(c,c)	Q(a)	¬Pc(g(c))
Q(a)	¬P(c,c)	P(c,g(c)			
¬Q(a)			¬Q(a)		

Esta regla enuncia en cierta forma, el principio de exclusión de los sistemas formales.

R3. Si se encuentra en el conjunto de cláusulas una cláusula que consta únicamente de una literal L, y además no existe otra cláusula que conste únicamente de la literal ¬ L se eliminan todas las cláusulas en las que aparece L (excepto la cláusula que consta únicamente de L) y todas las ocurrencias de ¬L.

Ej:

R3

Q(a)	P(c,g(c))			P(c,g(c))
P(c,c)	Q(a)	¬Pc,g(c))	P(c,c)	¬Pc,g(c))
¬P(c,c)	Q(a)	¬Pc(g(c))	¬P(c,c)	¬Pc(g(c))
¬Q(a)			¬Q(a)	

R4. En caso de que no se pueda aplicar ,R1, R2 ni R3, se escoge una literal L tal que aparezcan L y¬L en el conjunto de cláusulas, y se forman los siguientes conjuntos simplificados:

R4.1- El conjunto original menos las cláusulas en donde aparece L.
menos las ocurrencias de ¬L
más la cláusula unitaria: L.

R4.2- El conjunto original menos las cláusulas en donde aparece $\neg\neg L$
menos las ocurrencias de L
más la cláusula unitaria: $\neg\neg L$

En otro ejemplo:

$\neg R(a,b)$	$R(c,f(c))$
$R(a,b)$	$R(c,f(c))$
$R(a,b)$	$\neg R(c,f(c))$
$\neg\neg R(a,b)$	$\neg R(c,f(c))$
$\neg Q(a)$	

R4

Para L=R(a,b)

R4.1	**R4.2**
$R(a,b)$	$\neg R(a,b)$
$R(c,f(c))$	$R(c,f(c))$
$\neg R(c,f(c))$	$\neg R(c,f(c))$
$\neg Q(a)$	$\neg Q(a)$

En las dos ramas de este árbol, de demostración se puede ver por su simple forma, que no tiene modelo ninguna de las dos.
Y también para esta literal, en las dos ramas del árbol resultante podemos ver, por su simple forma que no tiene modelo, ninguna de las dos. Análogamente sucederá para L= $\neg R(c,f(c))$.

Este procedimiento se comporta como "algoritmo" siempre y cuando se le alimente con un conjunto finito de cláusulas instanciadas. La correctez y completez de este algoritmo está fundamentada por los siguientes teoremas [Loveland].

Teorema. Un conjunto finito de cláusulas instanciadas con términos del universo de Herbrand son satisfacibles si, y sólo si así lo indica el procedimiento Davis-Putnam.

Teorema. El procedimiento Davis-Putnam siempre termina en un número finito de pasos dando un resultado correcto, y las siguientes equivalencias lógicas:

R1) $A \vee P \vee \neg P \equiv A$
R2) $(A \vee B) \wedge A \wedge D \equiv A \wedge D$
R3) $L \wedge (A \vee L) \wedge (B \vee \neg L) \wedge D \equiv L \wedge B \wedge D$
R4) $(L \vee A) \wedge (\neg L \vee B) \wedge D \equiv (A \wedge D \wedge \neg L) \vee (B \wedge D \wedge L)$

Resolución.

El siguiente procedimiento es el de Resolución. Este es similar al anterior en cuanto a que se basa también en el teorema de Herbrand, que se alimenta con cláusulas y en que es un procedimiento de refutación, pero adquiere diferencias en ciertas partes fundamentales. Una de estas diferencias es la de permitirse usar una regla de inferencia (la regla Resolución) que no preserva la equivalencia lógica de un conjunto de cláusulas. De esta forma se permite que el modelo de un conjunto de cláusulas inferidas no lo sea de las cláusulas originales. Además, el sistema está formado por una sola regla de aridad 2. Esta última, la regla Resolución, se aplica repetidamente a diferentes pares de cláusulas llamadas **Cláusulas Padres** y, da como resultado una sola cláusula llamda **Resolvente**. La regla de inferencia Resolución es la siguiente:

Dadas las cláusulas:

$$\begin{array}{ll} A \vee L & \text{en donde L es una literal} \\ \underline{\neg L \vee B} & \\ A \vee B & \end{array}$$

Lo que es equivalente a lo siguiente:

$$\begin{array}{ll} \neg A \rightarrow L & \textbf{cláusulas padres} \\ \underline{L \rightarrow B} & \\ \neg A \rightarrow B & \textbf{resolvente} \end{array}$$

Posteriormente se incorpora a este sistema el Algoritmo de Unificación que permite comparar dos términos t_1 y t_2 para determinar si mediante alguna sustitución en las cláusulas en las que se encuentran, es posible la igualdad:

$$t_1 : s = t_2 : s$$

En donde s es la sustitución mencionada para las cláusulas en donde se encuentran t_1 y t_2 ; t_1:s y t_2:s denotan el resultado de efectuar la sustitución indicada en los términos t_i , i=1,2.

De esta forma se puede hacer deducciones como la siguiente:

$$\begin{array}{ll} \{A \vee P(t_1)\}s & \textbf{cláusulas padres} \\ \underline{\{B \vee \neg P(t_2)\}s} & \text{mediante la sustitución} \\ A \vee B & \textbf{resolvente} \end{array}$$

en donde A y B son proposiciones del cálculo de predicados. Mediante Unificación es posible trabajar con cláusulas sin instanciar para descubrir si dos literales son unificables.

Ejemplo:

P(X1,g(X1))	Q(a)	¬P(X1,g(X2))	cláusula 1
Q(a)	P(X2,g(X2))		cláusula 2
P(X3,X3)	Q(a)	¬P(X4,g(X4))	cláusula 3
¬P(X5,X5)	Q(a)	¬P(X5,g(X5))	cláusula 4
Q(a)	¬P(c,c)	P(c,g(c))	cláusula 5
¬Q(a)			cláusula 6

El procedimeinto de Resolución y el algoritmo de unficación quedan acoplados en 1968. Fue llamado Resolución General y consiste en buscar pares de cláusulas como el de las cláusulas 3 y 4:

P(X3,X3)	Q(a)	¬P(X4,g(X4))	cláusula 3
¬P(X5,X5)	Q(a)	¬P(X5,g(X5))	cláusula 4

de manera que contengan a dos literales contrarias en su signo y con la misma letra predicativa y cuyos subtérminos sean todos unificables por una misma sustitución s. Esta sustitución se pueden escoger de manera que sean lo más generales posibles, es decir, que contengan a la menor cantidad de términos instanciados con constantes o términos constantes.

En el ejemplo, las mejores sustitución es:

{ P(X3,X3)	Q(a)	¬P(X4,g(X4))	} X3/X8 X4/X8	cláusula 3
{ ¬P(X5,X5)	Q(a)	¬P(X5,g(X5))	} X5/X8	cláusula 4

Las cláusulas quedan:

P(X8,X8)	Q(a)	¬P(X8,g(X8))	cláusula 3
¬P(X8,X8)	Q(a)	¬P(X8,g(X8))	cláusula 4

Al aplicar Resolución con unificación se obtiene una nueva cláusula:

{ P(X3,X3)	Q(a)	¬P(X4,g(X4))	} X3/X8 X4/X8	cláusula 3
{ ¬P(X5,X5)	Q(a)	¬P(X5,g(X5))	} X5/X8	cláusula 4
Q(a)	¬P(X8,g(X8))		**resolvente:**	cláusula 7

Aquí se obtiene lo que se llama unificador más general con el cual es posible dejar algunas variables sin instanciar y así se facilita el proceso. El hecho de haber motido una de las dos Q(a) resultantes y uno de los P(X5,g(X5)) se llama **factorización** (*"factoring"*) y en algunos sistemas se incluye como una regla de inferencia más.
El objetivo de este procedimiento es encontrar las combinaciones de esta regla aplicada a cualquiera de las cláusulas que se tienen, a fin de obtener la cláusula vacia: []

Ejemplo:

{ Q(a)	P(X2,g(X2)) } X2/X5		cláusula 2
{Q(a)	¬P(X5,g(X5)) }		cláusula 7
	Q(a)	resolvente:	cláusula 8
	¬ Q(a)		cláusula 6
		resolvente:	cláusula vacía

Así, utilizando la cláusula 8 y sin necesidad de usar unificación, la cláusula 8 y la cláusula 6 nos llevan a la cláusula vacía, la cual es la meta de este procedimiento y significa que el conjunto original de cláusulas es insatisfacible. Es obvio que hay muchas formas posibles de aplicareste procedimiento, algunas veces de demostrará primero una fórmula porque la secuencia de aplicación es la adecuada.
En el ejemplo anterior, siempre se utiliza en cada aplicación de Resolución, una cláusula que es resolvente previos. A esta forma de aplicar Resolución se le llama **Resolución Lineal** y como ésta hay muchas variantes.

El boom de Resolución.

A partir de 1960, surge entre los investigadores de la DAT, una cierta convicción de que los métodos de manipulación sibólica representaban la opción deseada para construir un demostrador que fuera altamente eficiente y de mayor amplitud que los ya mencionados. Aumentaron los esfuerzos para encontrar, mediante el ingenio, herramientas conceptuales que resultaran más eficientes.

A principios de los sesentas, surge una de las ideas que tomaron mayor importancia en los trabajos subsecuentes. El Algoritmo de Unificaciòn o simplemente Unificación formó parte fundamenta en el desarrollo de la Demostración Automática. Fue desarrollado principalmente por J.A. Robinson, quien a partir de 1964, comenzó a trabajar en el camp asesorado por William F. Miller en Argonne National Laboratories en los Estados Unidos. Este, al analizar el problema de la explosión combinatoria de término en el algoritmo de Davis-Ptnam, se interesó en revisar las ideas propuestas por Dag Prawitz [Prawitz] en torno a la realización de búsquedas para la sustitución de términs en cláusulas. Es ahi donde Robinson encontró los antecedentes a las ideas de Unficación, (esta idea se puede encontrar también esbozada en otro artículo de 1930 de Hebrand.
Como se ha visto, en la manipulación simbólica de fórmulas (que involucran metodos de Herbrand), no todos los elementos del universo de Herbrand son importantes para la demostración. El algoritmo de unificación provee una forma de acercarse a las sustituciones adecuadas en las cláusulas pertenecientes a una fórmula a fin de evitar la enumeración de todos los símbolos posibles del universo de Herbrand. De esta forma, y en base a las ideas mencionadas, J.A. Robinson propone en 1963 un nuevo método de demostación automática que combinó con el algoritmo de Unificación (como lo nombró) y una derivación de la regla "cut" de Gentzen.
Este procedimiento fue llamado Resolución. Fue publicado en 1965 y paulatinamente atrajo la atención de los investigadores dedicados al campo de la demostración automática.
El método de Resolución con unificación tal como ha sido explicado en la sección anterior, no resultó muy eficiente en la práctica. Sin embargo, las ideas subyacentes en

él, fueron de gran importancia, com se verá. El uso de algunas restricciones o guías para la aplicación de esta regla en demostradores, pueden aumentar su eficiencia grandemente.
Diferentes grupos de investigación se sintieron atraídos hacia esta nueva propuesta y, casi de inmediato fueron adoptando las ideas de Resolución y tratando de mejorarlas surgieron algunas modificaciones como: "Unit Preference" o preferencia por Cláusulas Unitarias e Hiperresolución, ambas surgidas dentro del primer años de vida de Resolución.

Cláusulas de Horn
Una cláusula de Horn es una cláusula que tiene a lo más un átomo no negado, es decir, son de la forma:

$\neg A_1 \vee ... \vee \neg A_n \vee B$ ó $\neg A_1 \vee ... \vee \neg A_n$

que son lógicamente equivalentes a:

$(\neg A_1 \wedge ... \wedge A_n) \rightarrow B$ ó $\neg(A_1 \vee ... \vee A_n)$ que se denota: $(A_1 \vee ... \vee A_n) \rightarrow$

Las cláusulas de Horn se clasifican en:

$(\neg A_1 \wedge ... \wedge A_n) \rightarrow B$	**reglas** (con exactamente un átomo no negado: B)
$(A_1 \vee ... \vee A_n) \rightarrow$	**metas** (sin átomo no negado)
$\rightarrow B$	**hechos** (cuando n=0)
[]	**cláusula vacía** (es sólo una notación y significa fin de la demostración)

Utillizar este tipo de cláusulas, únicamente significa reducir el lenguaje a un subconjunto del Lenguaje de Primer Orden. No obstante lo anterior, este conjunto de cláusulas, parece ser suficientemente expresivo para la mayoría de las aplicaciones de la DAT.
Por otro lado, los métodos que utilizan Resolución se hacen muy eficientes al restringirse a cláusulas de Horn. Por ejemplo, el lenguaje de programación lógica Prolog del que se hablará posteriormente, utiliza cláusulas de Horn como su lenguaje de programación.

Resolución Unitaria

La variante de Resolución "Unit Preference" o "UR-Resolution" ("Unit Resulting Resolution), consiste en aplicar en primera instancia Resolución a las cláuslars que contan únicamente de una sola literal. Así, se garantiza que el resolvente es de una longitud menor que la cláusula base utilizada junto con la cláusula unitaria. Esta estategia de Resolución fue propuesta por Larry Wos, Dan Carson y George Robinson.

En su caso más general, se expresa así:

$$\frac{\begin{array}{l} A_1 \wedge \ldots \wedge A_m \rightarrow B_1 \vee \ldots \vee B_n \\ \rightarrow A_1 \\ \vdots \\ \rightarrow A_m \\ B_1 \rightarrow \\ \vdots \\ B_{n-1} \rightarrow \end{array}}{\rightarrow B_n}$$

Este método junto con el algoritmo de Unificación es completo bajo cláusulas de Horn.

Hiperresolución.

Otro ejemplo de un refinamiento de Resolución fue inventado por el mismo Robinson también en 1965, y es llamado Hiperresolución, ya que permite usar Resolución varias veces en una sola inferencia. Esta variante consiste en utilizar como base de resolución sólo cláusulas que contienen literales positivas (del estilo: A_i) y efectúa en un sólo paso todos los resolventes posibles con cláusulas que contienen a la negación de alguna de estas literales.

Estas variantes, como se podrá obervar, aceleran la velocidad en que se llega a una refutación mediante Resolución y presentan ventajas sobre ésta. Estos son sólo algunos ejemplos de refinamientos que se han desarrollado sobre este método de demostración automática. En el libro: *Automated Theorem Proving* (Loveland) se explican detalladamente 26 de estas variaciones que se han desarrollado.

Grupos como el de la Universidad de Edimburgo, las revistas "Computational Logic", dirigida por Bernard Meltzer y la "Machine Intelligence" dirigida por Donald Mitchie, también en Edimburgo; el grupo de la Universidad de Marsella en Francia, dirigido por Alain Colmerauer, la Universidad de Rice, los Argonne National Laboratories y la Universidad de Stanford, estudiaron y propusieon nuevas ideas en torno a Resolución que evolucionaron y ganaron partidarios hasta formar lo que podríamos llamar el Boom de Resolución que siguió consolidándose hasta convertirse en el procedimient de demostración más importante en la DAT.

Otra de las aplicaciones a que dió fruto Resolución fue el desarrollo de la programación lógica. Esta consiste en una forma de programación que combina el tratamiento lógico semático de los problemas con los algoritmos computacionales adecuados para una máquina serial, es decir, cumple perfectamente con la definición que hemos dado de un demostrador automático.

Para finales de los 60, dos de los grupos interesados por Resolución llegaron a desarrollos interesantes: el de inteligencia artificial de la Universidad de Marsella, dirigido por Alain Colmerauer, Phillipe Roussel y Bob Pasero estaba interesado en crear sistemas de preguntas-respuestas que utilizaron el lenguaje natural y el de la Universidad de Edimburgo dirigido por Robert Kowalski. Estos grupos combinaron su trabajo con el estudio de las ideas desarrolladas en ese momento en torno a Resolución e implementaro una nueva variante de Resolución llamada SL-Resolución. La culminación de este trabajo fue un novedoso lenguaje de programación llamado Prolog que se aplica a un conjunto restringido del cálculo de predicados (Cláusulas de Horn) y que utiliza la sofisticación algoritmica de Resolución para ser un procedimiento eficiente de demostración automática. Este lenguaje rompe con la forma tradicional (imperativa) de los lenguajes de programación h ha demostrado ser excelente para la manipulación no numérica de datos como el procesamiento del lenguaje natural.

Otras Tendencias

Algunas de las ideas que se desarrollaron dentro del campo y que han tomado forma como métodos establecidos, pueden ser agurpadas en lo que llamamos la rama "Simulación del pensamiento humano o natural", en la DAT. Estas técnicas son prolíficas, como se verá posteriormente, en el periodo de fines de los sesenta en adelante. Aunque algunos de estos procedimientos de demostración carecen del rigor de la lógica, entiéndase que pueden no ser completos en el sentido lógico o que no incluyen a todo el lenguaje de primer orden: muchas veces resuelven con gran eficiencia problemas de complejidad mayor, que a otros demostradores más formalistas les constaría gran cantidad de tiempo de memoria de cómputo. El uso de estos métodos es de gran utilidad, por lo que trataremos de comprender algunos de ellos.

Haremos notar que estos procedimientos no son de demostración puramente simbólica, ya que al tomar en cuenta propiedades o resultados que se relacionan con el problema atacado, se está considerando la interpretación semática, los modelos o las porpiedades de una teoría axiomática. En estos casos las ayudas para el demostador que llamaremos heurísticas, consideran el significado de lo que se va a demostrar.

Por ejemplo, la fórmula demostrar es:

$$A_1 \wedge A_2 \wedge A_3 \rightarrow T$$

una heurística tomará en cuenta las propiedades expliícitas que posee la teoría generada por A_1, A_2 y A_3 para así encontrar una refutación de:

$$A_1 \wedge A_2 \wedge A_3 \wedge \neg T$$

En cierta forma, estas heurísticas implican la acción de un supervisor u observador por fuera del procedimiento de demostración que dirige el empleo de estos métodos.

Otra estrategia, es la de sacar propiedades que están implícitas en el conjunto axiomático de una fórmula y hacerlas explícitas al convertirlos en reglas de inferencia del procedimiento de demostración. Este es el caso de las Reglas de Sustitución (en inglés, "*rewrite rules*") utilizadas en las teorías que usan un lenguaje con igualdad. En estas ciertas igualdades, de entre las que se encuentran en los axiomas, son tomados el procedimento como reglas de ejecución del mismo. Un ejemplo de reglas de sustitución que pueden ser incorporadas en la teoría de grupos son:

$x + (-x) \rightarrow 0$ neutro aditivo

$((x+y)+z)=(x+(y+z))$ propiedad asociativa

Esta corresponde a un axioma de la misma teoría, y es implementada como una regla del procedimiento de demostración.

Además, se puede intenetar el transformar mediante estas reglas a todos los términos a una forma canónica, de manera que todos los término que aparecen en las instancias de una fórmula, son más simples y en donde la verificación de su validez es más sencilla. En el ejemplo anterior, se reducen todos los términos que son de la forma $+(x,-(x))$ a 0 que es un término más simple. Hay que hacer notar que no siempre se puede llegar a una forma normal de los términos, y por lo tanto no es un método totalmente efectivo, aunque existen sistemas axiomáticos en los que sí se tiene completud. Como ejemplo de esta heurística, podemos citar el algoritmo del Pressburger, para la aritmética de la suma $\langle N,+\rangle$ atacado por Martin Davis [Davis]. Esta misma idea la podemos encontrar en una variante de Resolución llamada Demodulación o Reducción para el manejo de la igualdad.
Demodulación:

$$\frac{C[r] \qquad \rightarrow (r \approx t)}{C[t]}$$

cláusula en donde aparece r como término

Estas reglas surgen alrededor de 1970 y son adoptadas por varios investigadores como Knuth, Benedix obteniendo buenos resultados en la práctica.

Una idea de A. Robinson mencionada anteiormente es la de considerar como apoyo a la demostración a un modelo de los axiomas de la fórmula. Como ya se vió, se puede obtener una guia de lo que es necesario hacer para la refutación de estos axiomas junto con la negación de la conclusión. Esto puede ser útil para eliminar caminos erróneos de demostración causados por tomar en cuenta modelos en los que los axiomas no se cumplen, (antecedentes inconsistentes implican cualquier conclusión). Se Instuye que la contradicción o inconsistencia de una fórmula se encuentra en los axiomas junto con la negación de la conclusión y no en los axiomas solos. Además, considerar un modelo real (completo) de los axiomas de una fórmula inconsistente, debe ayudar a encontrar la refutación de ésta.

Por otro lado, una nueva mejora para un demostrador es la de contar con procedimiento que trabaje en mayor nivel (metanivel), que pueda realizar diferentes labores y tomar otras decisiones de un procedimiento más mecánico o de bajo nivel. Las decisiones que puede realizar este "supervisor" pueden ser, por ejemplo, el suspender una rama de demostración de un procedimiento meánico cuando ésta es demasiado larga en el tiempo o se tiene "conocimiento" de que no conduce a nada. o la de aplicar ciertas reglas en algún orden para hacer más rápida la demostración; o el de realizar ciertas sustituciones convencientes en una fórmula. Los supervisores a su vez, pueden ser reglas que se aplican mecánicamente según un metaprocedimiento. También puede consistir de relgas heuristicas introducidas por un usuario e inclusive pensar en la posibilidad de que el supervisor sea el mismo usuario encargado de demostrar cierta fórmula con uso de conocimiento matemático para tomar las decisiones antes mencionadas. Dentro de esta linea podemos encontrar un demostrador llamado SAM ("*Semiautomated Mathematics*"). Éste comenzó siendo un verificador de demostraciones y continuó creciendo hasta convertirse en un sistema de interacción con el matemático. En 1967 llegó a demostrar, con la supervisión de un matemático, un lema de Teoría de Latices reconocido como un avance hacia una pregunta abierta [Loveland].

Otra estrategia de ayuda a la demostración automática, es el empleo de cierta base de conocimiento y del reconocimiento de patrones en fórmulas. Consisten en tomar como ejemplo f'ormulas similares o análogas demostradas previamente y que, por lo tanto, siguiendo un razonamiento similar, pueden ser demostradas. Como ejemplo, tomemos el "*Geometry Theorem Prover*" de Gerlenter, ya mencionado. Este puede resolver mediante un mismo razonamiento, varias pruebas, gracias a que en Geometría se encuentran fácilmente simetrías en diferentes problemas.

Es útil también, buscar ciertas simetrías y analogías entre fórmulas de una manera puramente sintáctica. Esto, sin embargo, involucra también el uso de otro sistema que descubra estas simetrías o analogías actuando sobre el demostrador.

Como refinamiento más allá de los mecionados, se puede dar todos aquellos utilizados por la Inteligencia Artificial, en gran variedad. El término "Base de Conocimiento" se refiere a un conjunto de datos que contiene hechos o fórmulas relacionados con cierto problema y que ha sido demostrados previamente o se incluye como ayuda para la demostración de un teorema. Esta idea se sigue de la necesidad de un matemático de utilizar conocimiento previo sobre un tema, como lo son, por ejemplo, lemas y otros teoremas, para demostrar una fórmula: Los problemas a los que uno se podría enfrentar, al tratar de implementar ayudas como ésta, pueden ser, el escoger un teorema previo de entre una base abundante de otros teoremas para utilizarlos y el de cómo utilizarlos. También es un problema a resolver, cuáles resultados obtenidos por un demostrador son importantes y por lo tanto, tendrán que formar parte de la base de datos. Con todos estos problemas es necesario encontrar soluciones sencillas, de implementar y que sean claras de entender. En otro caso podríamos llegar a los extremos de una base de conocimiento tan grande que sea impráctica de usar y, por otro lado, una base de conocimiento inútil o demasiado corta.

CAPITULO III

Un procedimiento para demostración automática

En este capitulo se presentará un procedimiento de demostración automática basado en el procedimiento Davis-Putnam (antes mencionado). A éste se le alimenta con un conjunto de cláusulas sin instanciar, o sea que se presuponen hechas las transformaciones del capitulo 1. Para hacer esto, automáticamente se pueden utilizar los programas: [Sabre], [Murguia].

El procedimiento, como se puede ver en el Diagrama 2, contiene varios procesos que atacan el problema desde puntos de vista diferentes y contiene además un módulo de interacción con el usuario.

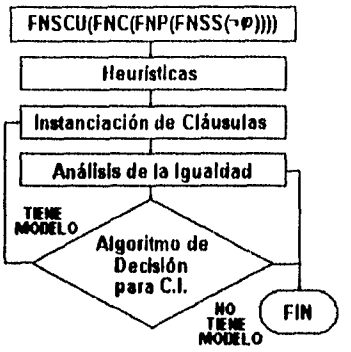


diagrama 2

Los módulos de que está formado son:

- Heurísticas
- Instanciación de Cláusulas (Instancias de Sustitución dadas para el usuario)
- Análisis de la Igualdad
- Algoritmo de decisión pára cláusulas instanciadas sin igualdad (Algoritmo Davis Putnam)

La fórmula que se quiera demostrar, primero pasará por el módulo de heurísticas. Este detectará asi la fórmula pra la que se quiere insatisfacibilidad, tiene modelo. Los siguientes tres módulos se repiten hasta que se logran agotar las posibilidades de que la fórmula tenga modelo. Esto último se hace añadiendo al conjunto de instancias de la fórmula (en un principio vacio) nuevas instancias de sustitución de la fórmula a demostrar.

Nótese que el procedimiento anterior puede no terminar nunca, es por esto que no llamamos algoritmo a nuestro método de demostración automática . No se garantiza

que siempre llegue a un fin. Sin embargo, el Teorema de Herbrand garantiza que la fórmula es universalmente válida si y sólo si el procedimiento así lo indica.

Nótese que para cualquier fórmula α FNSS(α) es siempre una fórmula universal pura, es decir, de la forma $\forall x_1 \ldots x_n \beta$ con β sin cuantificadores.

Se manejarán en este capítulo los siguientes dos ejemplos a fin de que se pueda seguir el procedimiento completamente:

Sean

Ej. A: $\varphi = \left(\forall x \exists y \left(Pxy \vee Pyx\right) \wedge \forall x \forall y \left(Pxy \rightarrow Pyy\right)\right) \rightarrow \exists z Pzz$

Ej. B: $\varphi =$

$$\forall x \forall y \left(g(x) \approx y \wedge \left(\neg Py \vee Pf(x)\right) \wedge Rx \wedge Py\right) \rightarrow \forall z \exists w \exists t \exists u \exists v \left(z \approx u \rightarrow \left(Pw \wedge Rw \wedge v \approx z \wedge t \approx z\right)\right)$$

las fórmulas a demostrar.

Estas al ser negadas y convertidas a su forma clausular quedan como sigue:

Ej. A: Forma Clausular de ψ = FNSS($\neg\varphi$)

Pxf(x) ∨ Pf(x)x
¬Pwy ∨ Pyy
¬Pzz

Ej. B: Forma Clausular de ψ = FNSS($\neg\varphi$)

g(w) ≈ 1
b ≈ u
¬Py ∨ Pf(x)
Rz
Pr
¬Ps ∨ ¬Rs ∨ ¬v≈b ∨ ¬t≈b

para probar en éstas, Ya no su validez universal sino su insatisfacibilidad.

Otro ejemplo:

Ejemplo C:
Px ∨ Pf(x) ∨ Qy ∨ ¬Qz ∨ x≈y
Pa ∨ Qw ∨ Qr a≈w
¬Pf(t) ∨ ¬Qa ∨ ¬a≈t

Heurísticas.

El objetivo de estas reglas es el de eliminar casos triviales, en los que ψ tiene modelo, es decir, la fórmula original φ no es universalmente válida y por lo tanto, el

procedimiento siguiente no terminaria Cuando la fórmula ψ sea tal, que por su forma (aún sin instanciar sus variables), siempre tenga modelo, estas heuristicas lo detectan. Se evita, de esta manera, caer en el error de intentar demostrar algunas fórmulas que no son universalmente válidas. Las heurísticas que se proponen son las siguientes:

A. Si en cada cláusula hay por lo menos una igualdad, entonces, la fórmula ψ tiene modelo.

Dado este caso, representando a en la Forma Normal Disyuntiva (dual de la F.N. Conjuntiva o Clausular) de ψ, podemos encontrar una subfórmula de que sea una conunción de igualdades y que esté separada del resto por el conectivo "$\vee$". Esta subfórmula siempre tiene modelo, pues siendo conjunción de igualdades, basta tomar un universo unitario, es decir con un solo individuo e interpretar como ese único individuo a todas las constantes que pudieran aparecer en la subfórmula. Entonces todas las igualdades serán verdad en esa interpretación para todas las instancias de las variables que pudieran aparecer (cuantificadas universalmente).

La razón esencial por lo que esto sucede es que las afirmaciones de tal subfórmula son <u>todas no negadas</u> y por lo tanto ψ siempre tendrá modelo. En esta situación, podemos concluir que la fórmula original φ <u>no</u> es Universalmente Válida y terminar el proceso.

B. Para que un conjunto de cláusulas no tenga modelo, es necesario que contenga una cláusula que tenga exclusivamente literales positivas y una cláusula que tenga exclusivamente literales negativas.

Tomemos el caso contrario, es decir: que en toda cláusula de ψ podamos encontrar una literal positiva o bien que en toda cláusula de ψ podamos encontrar una literal negativa. En tal caso tendremos una subfórmula de ψ, en su forma normal disyuntiva, que consiste únicamente de literales positivas o una que consiste únicamente de literales negativas, unidas por conectivos "$\wedge$" y unida con el resto de ψ por un conectivo "$\vee$". En cualquiera de los dos casos esta subfórmula siempre tendrá modelo ya que si es una conjunción de literales positivas es posible dar una interpretación del tipo de Herbrand que sea modelo; y si es una conjunción de literales todas negativas, el complemento del modelo de Herbrand anterior, es modelo suyo. Por lo tanto, la fórmula tendrá modelo ya que la subfórmula mencionada está conectada al resto por una disyunción. Al igual que el caso anterior podemos afirmar que la fórmula ψ siempre tendrá modelo, el proceso termina y φ <u>no</u> es U.V.

La justificación anterior está basada en los siguientes teoremas:

Teorema. Si S es un conjunto de cláusulas, S tiene modelo si y sólo si S tiene modelo de Herbrand.

(Este teorema no siempre es cierto para fórmulas que no están en forma clausular)

Teorema. Todo conjunto de cláusulas donde toda cláusula tenga al menos una literal positiva o donde toda cláusula tenga al menos una literal negativa, tiene modelo de Herbrand. Véase [Lloyd].

C. Si para algún simbolo de predicado P, no hay una cláusula que contenga literales con P, únicamente positivas y otra cláusula en donde haya literales con P, únicamente negativas, entonces es posible eliminar todas las cláusulas que contengan el predicado P.

En caso contrario, tendríamos, de la misma manera que la justificación de la regla naterior, varias subfórmulas de φ que contienen proposiciones con P sólo positivas (o sólo negativas). Las cláusulas que contienen a P no son significativas para la demostración de que la fórmula tenga o no modelo desde el punto de vista del método empleado, por lo que se pueden eliminar y el resultado es un conjunto de cláusulas equiposible lógicamente.

D. Si existe una función que aparece en una y sólo una cláusula y dentro de esta cláusula hay otra función, que también aparece sólo en esa cláusula de la misma aridad, entonces es posible identificar estas dos funciones. Ver [Prawitz]

Es claro que cualquier interpretación para una de las funciones se les puede dar a la otra, sin alterar el hecho de tener o no tener modelo. Es decir, son lógicamente equiposibles.

Instanciación de Cláusulas.

El paso siguiente, según el plan que se ha trazado para la demostración de φ (o refutación de ψ=FNSS($\neg\varphi$) en FNC, será instanciar las variables de ψ con términos constantes. Estos pueden ser:

- Constantes ó
- Funciónes aplicadas a términos constantes
  (su definición, como se vió es recursiva).

Al resultado de hacer esto lo llamamos instancia de sustitución de una cláusula.

Por el teorema de Herbrand, el cojunto ψ no tiene modelo si y sólo si existen instancias de sustitución de ψ: $\psi_1,\ldots,\psi_n$ tales que:

$$\psi_1\wedge\ldots\wedge\psi_n \quad \text{no tiene modelo.}$$

Una vez teniendo un conjunto de instancias de sustitución es sencillo comprobar si éste tiene o no modelo. Haya, entonces, una razón por la cual los demostradores de teoremas no se han podido llevar a niveles prácticos, el número de posibles modelos para un conjunto de cláusulas instanciadas, presenta el siguiente índice de crecimiento:

$\prod|C_i|^n$ donde $|C_i|$ es el número de literales en la cláusula C_i y n es el número de instancias que se han efectuado hasta ese paso.

Esta cantidad, como se puede observar, tiene un crecimiento exponencial que, en pocos pasos saturaría cualquier máquina en donde se intente implementar. El tiempo requerido para analizar un número así, lo vuelve impráctico aún con la mejor

computadora existente. Es por esto que se desea que las instancias que se utilicen en el procedimiento sean las adecuadas y sólamente éstas. De esta manera, el procedimiento puede hacerse tan efectivo como se desee.

Por otra parte, hasta donde alcanza la revisión hecha para este trabajo, no se ha encontrado un procedimiento efectivo para escoger las instancias (es decir, que no gaste demasiado tiempo o memoria). Sin embargo, la experiencia humana, de la misma forma que cuando se encuentra la demostración de un teorema, se puede considerar la mejor de las opciones para optimizar el proceso.

Es por esto, y siguiendo el consejo de Hao Wang [Wang] de hacer demostradores interactivos, que hemos preferido hacer que esta parte del procedimiento sea llevada a cabo por el usuario; en ausencia de algún algoritmo o criterio efectivo que nos indique qué sustituciones hacer.

Se pasará, pues, por este módulo de interacción con el usuario tantas veces como el problema y el proceso siguiente lo indiquen. Se espera de éste que realice sustituciones adecuadas, usando su intuición de manera que el procedimiento sea eficiente.

Ej A: Primera instancia = {w/c, x/c, y/c, z/f(c)}

Pcf(c)	Pf(c)c
¬Pcc	Pcc
¬Pf(c)f(c)	

Ej. B: Primera instancia = { l/a, r/g(b), s/a, t/f(f(a)), u/f(f(a)), v/a, w/b, x/a, y/f(a), z/a}

		g(b)≈a		
		b≈f(f(a))		
Pf(a)	¬Pf(a)			
Ra				
Pg(b)				
¬Pa	¬Ra		a≠b	f(f(a))≠b

Ej. C: Primera instancia = {x/a, y/f(b), z/a, w/f(b), r/f(b), t/a}

Pa	Pf(a)	Qf(b)	¬Qa	a≈f(b)
Pa		Qf(b)	Qf(b)	a≈f(b)
¬Pf(a)		¬Qa	¬a≈a	

Análisis de la Igualdad

Dadas ciertas instancias de sustitución de ip, tomamos de las cláusulas que contienen igualdades, un conjunto de éstas, con ellas se realizarán una serie de algoritmos que pueden dar como respuesta que cierto conjunto de cláusulas instanciadas de sustitución en particular tiene modelo o no; pero exclusivamente por las igualdades que aparezcan en ella.

Este tratamiento de las instancias se divide en tres algoritmos:

1. Algoritmo de ordenamiento del conjunto de igualdades
2. Algoritmo de ejecución de clases de equivalencia
3. Análisis de desigualdades

que se efectúa de la siguiente forma:

Dado un ordenamiento de los conjuntos de igualdades, se le aplica a cada conjunto, un tratamiento especial para la igualdad. Este algoritmo se alimenta con un conjunto de cláusulas instanciadas y un conjunto de igualdades dentro de estas instancias.

El diagrama de flujo de esta parte del demostrador se encuentra esbozado en el diagrama 3:

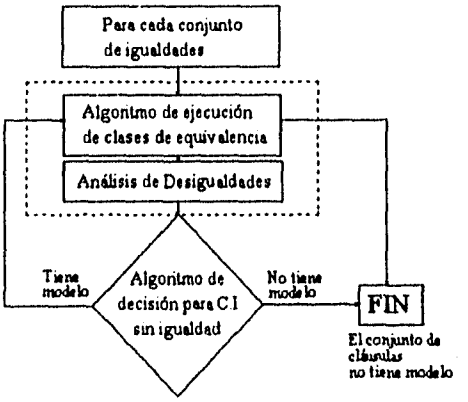


diagrama 3

Algoritmo de ordenamiento del conjunto de igualdades.

Empezamos por el ordenamiento mínimo posible, según un orden determinado. El orden que se utilizará para las igualdades es el siguiente:

Sean:

$$\begin{matrix} I_{11} & I_{12} & I_{1n_1} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ I_{m1} & I_{m2} & I_{mn_m} \end{matrix}$$

las igualdades en las cláusulas $C_1,\ldots,C_m$ respectivamente

Sabemos de antemano por la heurística A de la sección anterior, que no puede haber igualdades en cada cláusula, ya que esto nos llevaría a que la fórmula ψ tiene modelo y φ no es U.V. Puede suceder, sin embargo, que haya cláusulas que únicamente contengan igualdades. Colocaremos a éstas últimas cláusulas en los primeros renglones de igualdades.

Con las cláusulas que contienen sólo igualdades se forman conjuntos mínimos de igualdades, de manera que contengan una y sólo una igualdad de cada una de estas cláusulas. Estos conjuntos también se ordenan: $M_0, M_1, \ldots, M_k$.

El primer conjunto consiste de las primeras igualdades de las cláusulas que sólo contienen igualdades (conjunto mínimo de igualdades).

Se forman los siguientes conjuntos:

$I_{00} = M_0, I_{10} = M_1$ en general $I_{j0} = M_j$

obsérvese que si no hay cláusulas que contengan sólo igualdades M = ϕ es el único conjunto mínimo.

En los ejemplos, el B tiene como único conjunto mínimo a $M_0 = \{g(w) \approx l, b \approx u\}$

Ahora se definen en los siguientes conjuntos de igualdades como sigue:

$I_{j_0} = M_j$ (el j-ésimo conjunto mínimo de igualdades)

$I_{j_1}, I_{j_2}, \ldots, I_{k_j}$ es una enumeración de todos los conjuntos que incluyen al conjunto mínimo M y que tienen a lo más una igualdad, de cada una de las cláusulas que tienen igualdades además de predicados, considerando todas las combinaciones o modos posibles de cumplir lo anterior.

De esta manera, al final obtenemos un orden para conjuntos de igualdades:
$I_{00}, I_{01}, \ldots, I_{0n_0}, I_{10}, I_{11}, \ldots, I_{1n_1}$, etc.

Estos conjuntos I_{jk} tienen propiedad de tener a lo más una igualdad de cada cláusula que tiene igualdades y predicados, y exactamente una igualdad de cada cláusula que únicamente tenga igualdades; y son todos los que se pueden formar así con las cláusulas instanciadas dadas de ψ.

En el ejemplo B, sólo hay un conjunto de igualdades y es el conjunto mínimo
I_{00} = {g(b)≈a,b≈f(f(a)}

En el ejemplo C:
I_{00} = {}
I_{01} = {a≈f(b)}

•según el orden utilizado aquí [Shostak]

Inmediatamente después se ejecuta el siguiente algoritmo

Algoritmo de ejecución de clases de equivalencia.

Para cada uno de estos conjuntos de igualdades de igualdades. Esto último sólo para los términos que se están considerando, es decir, aquellos que aparecen en las instancias de cláusulas que se están analizando. Dado un conjunto de igualdades I:
$I_{11} \approx I_{12} \wedge I_{21} \approx I_{22} \wedge \ldots \wedge I_{n1} \approx I_{n2}$
donde todas las I_{i1}, I_{i2} con i = 1,2... n son términos constantes, sea T el conjunto de términos que ocurren en alguna igualdad de I. El problema de decidir si un términmino I_k en este conjunto es igual a otro término I_l, puede solucionarse mediante el siguiente procedimiento.

Dado el conjunto de términos T, se realiza una partición $\{T_i, i \leq n\}$, de la siguiente manera:

1. Se toma el conjunto T de términos constantes y se considera cada elemento como una clase (ordenados por complejidad de términos y dos de igual complejidad, por orden lexicográfico).

Ej. B: I_{00} = {g(b)≈a, b≈f(f(a))} T = {a,b,g(b),f(f(a)))
T_1 = {a} T_2={b} T_3={g(b)} T_4 = {f(f(a))}

2. para cada ($I_{ik} \approx I_{jk}$) en el conjunto de igualdades, se une la clase de T_i con la de T_j para formar una nueva clase.

Ej. B: T_1 = {a,g(b)} T_2 = {b,f(f(a))}

3. Si $I_i \in T_i$ y $I_j \in T_j$ entonces $T_i \cup T_j = T_i$, es decir, se renombra a una de las clases como la unión de las dos.

4. Siguiendo un orden de complejidad determinado, se comparan $f_i''(\mu_1,\ldots,\mu_n)$ con $f_i''(\nu_1,\ldots,\nu_n)$ si es que existen dos térmios con el mismo símbolo funcional f_i''.

En caso de que para todo i $\mu_i = \nu_i$, es decir, pertenezcan a la misma clase de equivalencia correspondiente a estos dos términos

Aplicando los pasos 1 y 2 y luego los pasos 3 y 4 tantas veces como sea necesario y siguiendo el orden dado hasta terminar la lista de términos, obtendremos la particiión de T en clases de equivalencia obligadas por el conjunto I de igualdades.
Así, termina el algoritmo dando como respuesta un representante de cada clase de equivalencia y cuáles términos pertenecen a cada clase de equivalencia.

Para su implementación se utilizan las funciones UNION (X Y Z) y FIND(X). Véase [Tarjan]. Según éste, el número máximo de pasos a realizar para terminar es n, donde n es el número de términos de T. Esto, en tiempo de computadora, representa un proceso de crecimiento lento y por lo tanto resulta eficiente para nuestros propósitos.

Sustitución de términos equivalentes por representantes:
La idea es considerar el caso de que las igualdades de I_{jk} son ciertas, lo cual es equivalente a sustituir todos los términos iguales por un representante según las clases de equivalencia obtenidas y no considerar ya las igualdades:

Dado un conjunto de instancias de ψ y dado I_{jk} un conjunto de igualdades de C, definidos C_{jk} como C menos las cláusulas que tienen alguna igualdad en I_{jk} y menos las igualdades que no ocurran en I_{jk} sustituyendo en las literales, cada término por su representante según las clases de equivalencia obtenidas.

Análisis de desigualdades.
Ahora tomando en cuenta sólo a las cláusulas cuyas igualdades no están en I_{jk} se procede a un análisis de desigualdades:
En esta parte se pueden eliminar cláusulas y ocurrencias de igualdades; de manera que terminando este proceso, podamos llegar al siguiente paso, únicamente cláusulas sin igualdades (ni desigualdades) para ser analizadas. Dentro de este análisis, como ya se mencionó, sólo intervendrán cláusulas que no contengan alguna igualdad que pertenezca al conjunto I_{jk}. Una vez que se ha llevado a cabo el algoritmo de equivalencias y que se han sustituido los términos del conjunto de instancias por aquellos que son iguales en cualquier modelo de I_{jk}, es posible encontrar una refutación de C_{jk} observando las desigualdades de este conjunto. Llamaremos desigualdades imposibles para I_{jk} a aquellas que, después de haber hecho las sustituciones de términos relativas a I_{jk}, el resultado es una expresión del tipo $\neg(t \approx t)$ donde las dos ocurrencias de t son exactamente el mismo término. Es decir, estas literales son siempre falsas ya que para cualquier interpretación, un mismo término se interpreta igual.

Cualquier otra desigualdad es una desigualdad posible ya que puede tener una interpretación en la que sea cierta.

Una refutación de C_{jk} mediante desigualdades, consiste de una cláusula $C_i \in C_{jk}$ que está formada exclusivamente por desigualdades imposibles para I_{jk} (Hay que recordar que estamos analizando el caso en que I_{jk}, se cumple, para llegar a una refutación).

Si este no es el caso, podemos continuar buscando una refutación de C_{jk}, pero ahora sólo mediante símbolos predicativos.

Por otra parte, las cláusulas que contengan desigualdades posibles para I_{jk}, es decir, aquellas que no son necesariamente falsas para I_{jk}, tienen modelo siempre, pues siempre podemos elegir un modelo en el que los términos mencionados sean diferentes. Espresaremos lo anterior con la siguiente equiposibilidad lógica, denotada $\overset{\blacksquare}{\approx}$.
Si C es una cláusula del tipo anterior, entonces:

$$I_{jk} \wedge C_n \wedge C \overset{\blacksquare}{\approx} I_{jk} \wedge C$$

Entonces, podemos no tomar en cuenta, para el análisis subsecuente, aquellas cláusulas que contengan desigualdades posibles.

Ej B:
I_{00} = {g(b)≈a,b≈f(f(a))}

C_{00}	Pf(a)	¬Pf(a)		
	Pa	Ra		
{	¬Pa	¬Ra	a≠b	b≠b} *

Obsérvese que a≠b es una desigualdad posible, por lo que la cláusula entre llaves (*), no se toma encuenta en lo que sigue, y se vuelve a considerar en otra posible nueva instancia.

Por lo tanto, los pasos a seguir en este análisis de desigualdades son: Detectar si existe una cláusula que contenga sólo desigualdades imposibles.
En caso afirmativo C_{jk} no tiene modelo posible y por lo tanto el caso en que I_{jk} se cumple ha sido refutado. pasamos entonces a buscar una refutación para el siguiente conjunto de igualdades I_{jk+1} o I_{j+10}. En caso negativo, se elminarán todas las ocurrencias de desigualdades imposibles, se ignoran el resto de este procedimiento, pero no se eliminan.
Se tomarán en cuenta si se agrega una nueva instancia.
Como resultado obtendremos un conjunto de subcláusulas que no contienen igualdades ni desigualdades y que tienen modelo si, y sólo si el conjunto original C de instancias de cláusulas tiene modelo.

Ej. B:

Pf(a)	¬Pf(a)
Pa	Ra

Algoritmo de decisión para cláusulas instanciadas sin igualdad

Este algoritmo tiene como entrada un conjunto de cláusulas sin variables y sin igualdades C. A este conjunto le son aplicadas una serie de reglas **(R1...R4)** y criterios **(C1, C2 y C3),** basados en el procedimiento Davis-Putnam de refutación; véase [Davis Putnam], mediante el cual, es posible determinar si el conjunto de cláusulas tiene modelo o no.
Este grupo de reglas de simplificación y criterios de decisión, se utilizan dentro del algoritmo en la forma que indica el Diagrama 4.

Además en caso de que tengan modelo, el agoritmo convierte la fórmula en varias fórmulas más simples, (fórmulas hijas), dispuestas en un árbol, de manera que el conjunto original de cláusulas (raíz del árbol) tiene modelo si, y sólo si, alguna de éstas (hojas del árbol) lo tiene. Este proceso, entonces, además de indicar si la fórmula con que se alimenta tiene o no modelo, simplifica la fórmula para ser tratada posteriormente (en caso de tener modelo); y esto, sin que pierda su contenido, es decir, el conjunto de cláusulas aún junto con otro conjunto de instancias de cláusulas (C ∧ C') tendrá modelo si, y sólo si alguna de las fórmulas hijas ($C_i \wedge C_j$) (junto con C') lo tiene:

Las reglas que se aplican dentro de este algoritmo son las siguientes:

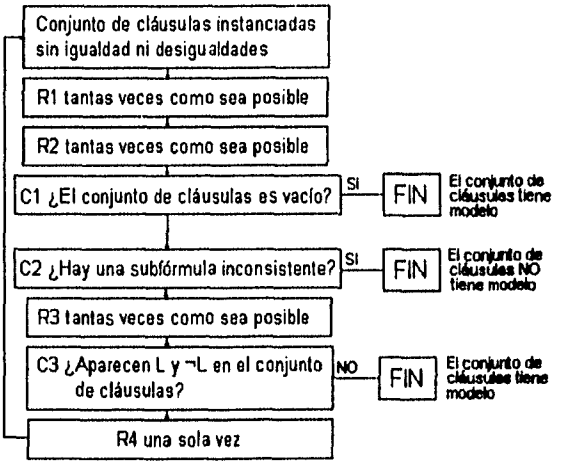


diagrama 4

R1. Se eliminan del conjunto de cláusulas todas aquellas que sean tautologías.

Estas serán todas aquellas cláusulas que contengan a una literal L y a su contraria en signo, denotada como ¬L

Ej. A:

Pcf(c)	Pf(c)c	R1	Pcf(c)	Pf(c)c
¬Pcc	Pcc	⇒		
¬Pf(c)f(c)				¬Pf(c)f(c)

Ej. B:

Pf(a)	¬Pf(a)		
Ra		R1	Ra
Pa		⇒	Pa

Esta regla se obtiene de la equivalencia lógica:

$$A \wedge P \vee \neg P \equiv A$$

Denotamos con **Ri** una aplicación de la regla.

R2. Si el conjunto de cláusulas hay dos cláusulas C_i y C_j tales que:

$$C_j = C_i \vee B$$

entonces se elimina la cláusula C_j

Esta regla se obtiene de la equivalencia lógica:

$$(A \vee B) \wedge A \wedge D \equiv A \wedge D$$

R3. En caso de que exista una cláusula con una sola literal L, y que además no exista otra cláusula con solamente la literal ¬L, se eliminan de C todas las ocurrencias de ¬L y todas las cláusulas que contengan a la literal L excepto la que sólo contiene a ésta.

Esta regla se justifica con la siguiente equivalencia lógica:

$$L \wedge (L \vee A) \wedge (\neg L \vee B) \wedge D \equiv L \wedge B \wedge D$$

Siempre y cuando B no sea una fórmula vacía.

R4. Si en una cláusula se encuentra una literal L, y en otra su negación ¬L (fuera de los casos mencionados anteriormente) entonces la fórmula original C es equiposible lógicamente a la disyunción de las fórmulas C_1 y C_2 definidas de la siguiente manera:

C_1 es el conjunto original de cláusulas menos las cláusulas que contienen a ¬L menos las ocurrencias de L.
C_2 es el conjunto original de cláusulas que contienen a L menos las ocurrencias de ¬L

La disyunción de las fórmulas así obtenidas, es equivalente lógicamente a la original mediante la aplicación de la siguiente equivalencia lógica:

$$(L \vee A) \wedge (\neg L \vee B) \wedge D \equiv (\neg L \wedge A \wedge D) \vee (L \wedge B \wedge D)$$

Con este paso se sistituye la fórmula C por dos fórmulas más simples.

Junto con las reglas, usamos algunos criterios para determinar si el conjunto de cláusulas tiene o no modelo. Estos criterios se explican a continuación.

C1. Si después de quitar las tautologías (R1), el conjunto C queda vacío, entonces C tiene modelo.
Además, podemos afirmar que la instancia de sustitución que se realizó no tiene ninguna utilidad para el procedimiento que se está empleando, por lo que la eliminaremos, ya que convierte a la fórmula ψ en una tautología.

C2. Si en el conjunto de cláusulas C se tiene una cláusula que consiste únicamente de una literal L, y otra cláusula consistente únicamente de la literal $\neg L$, entonces C no tiene modelo.

Este es el caso en que C es igual a $L \wedge \neg L \wedge B$, donde B es el resto de las cláusulas. Esto, evidentemente no tienen modelo posible.

C3. Si en la fórmula aparece una literal L en una cláusula y su negación $\neg L$ en otra cláusula, aplicamos la regla R4 obteniendo dos fórmula shijas mmás simples cuya disyunción es equiposible lógicamente a la original, y regresamos a repetir este mismo algoritmo para la primera fórmula hija, y luego para la segunda. Si en la fórmula ya no es posible encontrar literales L tales que su negación esté en otra cláusula, entonces la fórmula C tiene modelo. Esto sucede porque un conjunto de cláusulas en donde siempre ninguna de sus literales se encuentra negada en otra cláusula, siempre tiene modelo. Véase [Loveland]. Además podemos garantizar que la fórmula C se encuentra simplificada al máximo y es necesario entonces, realizar una nueva instancia de sustitución el el conjunto original de cláusulas, y agregársela a la fórmula C, junto con las instancias de igualdades y las cláusulas que no se tomaron en cuenta por tener igualdades posibles.

En resumen, los resultados que se obtienen alimentando este algoritmo con un conjunto de cláusulas instanciadas sin igualdades, son los siguientes:

1. En caso de que las cláusulas no tengan modelo, el algoritmo así lo indicará. Además, se tendrá al terminar de ejecutarse el algoritmo, un conjunto de varias cláusulas simplificadas que tendrán modelo si, y sólo si, junto con cualquier conjunto de cláusulas instanciadas tienen modelo.

2. En caso contrario, se obtiene una respuesta negativa indicadora de que se hizo una sustitución adecuada de las variables de la fórmula.
En ambos casos el algoritmo termna y su resultado es transferido al resto del procedimiento general de decisión.

Para ilustrar los pasos anteriores, terminaremos los dos ejemplos A y B que hemos desarrollado a lo largo de este capítulo:

Ej. A: Segunda instancia = {w/c, x/c, y/f(c), z/f(c)} $I_{00}=\phi$

La fórmula anterior, con la nueva instancia y las reglas usadas es:

Pcf(c)	Pf(c)c			Pcf(c)	Pf(c)c		Pf(c)c
¬Pf(c)f(c)	¬Pf(c)f(c)						
Pcf(c)	Pf(c)c	R2				R3	
¬Pcf(c)	Pf(c)f(c)	⇒		¬Pcf(c)	pf(c)f(c)	⇒	¬Pcf(c)
¬Pf(c)f(c)				¬Pf(c)f(c)			

Tercera instancia = {w/f(c), x/c, y/c, z/c} $I_{\infty}=\phi$

Pf(c)c				Pf(c)c			Pf(c)c
¬Pcf(c)		R2		¬Pcf(c)		R3	¬Pcf(c)
¬Pf(c)f(c)		⇒		¬Pf(c)f(c)		⇒	¬Pf(c)f(c)
¬Pf(c)c	Pcc			¬Pf(c)c	Pcc		Pcc *
Pcf(c)	Pf(c)c						
¬Pcc				¬Pcc			¬Pcc *

* Indica que no tiene modelo, por lo tanto ψ no tiene modelo y φ es universalmente válida.

Ej. B:
Segunda instancia = {l/f(b), r/f(f(a)), s/f(a), t/g(b), u/a, v/f(f(a)), w/a, x/a, y/g(b),z/f(a) }

g(b)≈a
b ≈ f(f(a))

		Ra			
Pa					
¬Pa		¬Ra		a≠b	f(f(a)) ≠b
Pf(a)	¬Pg(b)				
		Rf(a)			
Pf(f(a))					
¬Pf(a)		¬Rf(a)		f(f(a)) ≠b	g(b) ≠c

Aplicando el algoritmo de clases de equivalencia al conjunto I_{∞} de las cuatro igualdades anteriores, es fácil ver que se obtiene una sola clase {a,b,g(b),f(b),f(f(a))} , y sustituyendo todos por "a", vemos que las dos desigualdades son imposibles, por lo que obtenemos:

	Ra				Ra
Pa				Pa	*
¬Pa	¬Ra	**R2**		¬Pa	*
Pf(a)	¬Pa	⇒	Pf(a)		¬Pa
	Rf(a)	**R3**			Rf(a)
Pa		⇒			
¬Pf(a)	¬Rf(a)		¬Pf(a)		¬Rf(a)

De aquí, por el criterio C2, *indica que esta instancia C_{∞} no tiene modelo, por lo que, al no tener otro conjunto de igualdades que evaluar,la fórmula no tiene modelo y la fórmula original es universalmente válida.

Conclusiones

Como podemos observar, muchos, si no es que todos los métodos de demostración analizados utilizan el concepto de supervisor: un sistema que rige el comportamiento o los parámetros de otros sistemas de demostración y que se encuentra en otro nivel de ejecución. Esto nos lleva a tener que enfrentar los problemas de la complejidad de realización de estos metaprocedimientos propios de la Inteligencia Artificial.
Por otra parte, el desarrollo de la computación se ha ido incrementando hasta alcanzar niveles insospechados en los que grandes velocidades de cómputo y grandes capacidades de memoria son comunes aún para el usuario hogareño. A pesar de esto no se ha logrado un éxito total en demostraciones automáticas. Todo lo anterior nos hace pensar que es necesario buscar nuevos rumbos, diferentes opciones para lograr un avance más satisfactorio. Podemos encontrar ejemplos de buenos resultados surgidos a partir de la implantación de este tipo de métodos menos formales como algunos demostradores mencionados. Tal vez, en la actualidad se pueda hablar de un retorno a la búsqueda de métodos del estilo de imitación del pensamiento humano. O quizá combinaciones más complejas entre el estilo lógico y el llamado en inglés "*human oriented*".

Como conclusión, si bien podemos considerar a los métodos lógicos de demostración como los más adecuados para ejecutarse en una computadora (por ser procedimientos simples, seriales y deterministicos) probablemente sea más productivo considerar métodos no tan efectivos (en sentido lógico) pero algo más eficientes (no infalibles), pero útiles). Podemos encontrar ejemplos sofisticados de demostradores automáticos que precisamente combinan varios métodos (de los mencionados y otros) y que resultan ser herramientas útiles al matemático, al científico de la computación y que son de gran interés para la Inteligencia Artificial.

Apéndice 1: Manual

El propósito de este demostrador automático es el de facilitar el proceso de demostración simbólica, no el de realizarla totalmente. Es decir, éste proceso necesita de la interacción con el usuario en una parte de la demostración (elección de instancias).
Para los que no estén familiarizados con este método, lo podemos resumir, sin afán de dar una explicación completa, de la siguiente forma.

1. Sea A un conjunto finito de fórmulas de la Lógica de Primer Orden llamados axiomas.

$A = \{ A_1, A_2, \ldots, A_n \}$

y T otra fórmula también dentro de la Lógica de Primer Orden.
Uno se puede preguntar si:

$$A \vdash T$$

es decir, si T se deduce de A en el cálculo de predicados. Lo que es equivalente a que:

$$A \vdash T$$

(T es consecuencia lógica de A) y a que

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow T \text{ ----------} \varphi$$

sea una fórmula universalmente válida (U.V.)

Ejemplo: Sean A y T las siguientes fórmulas:
A = { f(x)≈x
¬Qa→Pyg(y)
¬(Qa ∨ Pf(z)z) →¬Pzw
¬(Qa ∨ ¬Ptf(v)) →¬Ptg(t) }

T = Qa

(Nótese que todas las variables aparecen libres en la fórmula. Este es un requisito para introducir las fórmulas al demostrador)

Se desea saber si la fórmula es U.V. O sea que:

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \wedge \neg T \text{ ----------} \neg\varphi$$

no tenga modelo, en donde ¬T es la negación de T.

2. Aún la fórmula no puede ser introducida en el demostrador. Para ello hay que hacerle una serie de transformaciones descritas en los capítulos I y II) hasta llegar a la Forma Clausular de ¬φ
Toda fórmula puede ser llevada asu forma clausular en la que se han eliminado los cuantificadores existenciales y los universales se omiten. Además, la fórmula se divide en cláusulas:

$$(\psi_{11} \vee \psi_{12} \vee \ldots \vee \psi_{1n})$$
$$(\psi_{21} \vee \psi_{22} \vee \ldots \vee \psi_{2n})$$
$$\vdots$$
$$(\psi_{m1} \vee \psi_{m2} \vee \ldots \vee \psi_{mn})$$

Esta última transformación se puede realizar automáticamente mediante un programa desarrollado en Prolog por Sonia Sabre: [Sabre] o por [Murguia] también en Prolog.

Ejemplo ψ ($\neg\varphi$):
f(x)≈x
∧ (Qa ∨ Pyg(y))
∧ (Qa ∨ Pf(z)z ∨ ¬Ptg(t))
∧ (Qa ∨ ¬Ptf(v) ∨ ¬Ptg(t))
∧ ¬Qa

Una vez hecho lo anterior, se puede introducir la fórmula en el demostrador:

Al ejecutar el programa mediante la instrucción: **A:\>dem.exe**, se cargará el programa a la memoria. Entonces aparecerá la siguiente pantalla:

Este menú al ser desplegado completamente se vería así:

las ventanas 1,2,3 son de Entrada de datos, Información y Procedimientos respectivamente.

Lo primero que hay que hacer es introducir la fórmula a la memoria temporal de la máquina. para esto hay que abrir la ventana de entrada y salida de datos.*Ent./Sal.* y de este menú escoger la opción *F. Origen.* Entonces aparecerá el cursor en la pantalla y se podra introducir la fórmula respetando la siguiente puntuación:

Para predicados: - Estos deberán representarse con una letra Mayúscula excepto la "X" que se encuentra reservada.
-No se ponen paréntesis para agrupar a los términos y éstos se separan con ",".

Pata téminos: -Como símbolos funcionales se pueden usar cualquier letra minúscula.
- Los subtérminos de n término se separan con comas.
- Todos los símbolos funcionales llevan paréntesis.

Para cláusulas: - Cada cláusula se escribe en un renglón separado.
- Los símbolos "v" se omiten y las literales (fórmulas atómicas o negaciones de fórmulas atómicas) se separan por "/".
-El símbolo "¬", se escribe Alt-6 en los teclados latinoamericanos y con Alt y el número: 170 directamente desde cualquier teclado. Inmediatamente después va el símbolo de predicado.

Las constantes pueden ser cualquier letra minúscula y las variables son: X1, X2, etc.
Al final de la fórmula, en un renglón aparte, debe ponerse únicamente un punto: "."

La fórmula del ejemplo queda correctamente escrita así:

(1) =f(X1);X1
(2) Qa/PX2:g(X2)
(3) Qa/Pf(X3);X3/¬PX3;X4
(4) Qa/¬PX5;f(X6)/¬PX5;g(X5)
(5) ¬Qa
(6) .

Una vez hecho esto, es necesario hacer instancias de sustitución en las cláusulas con el fin de que en conjunto sean inconsistentes. para hacer esto hay que entrar a la ventana: *Ent./Sal. - Instancia* y escribir con la puntuación correcta (la mencionada anteriormente) los términos constantes que sustituirán a las variables de cada cláusula. Se cambia de fórmula con las flechas. Después de hacer la instancia, teclear "*ESC*" para la opción de incluir dicha sustitución (S/N).

Ejemplo:

(1) =f(c);c X1/c
(2) Qa/Pc:g(c) X2/c
(5) ¬Qa

Después hay que correr el Algoritmo de Decisión para Cláusulas Instanciadas con las instancias que se tienen hasta el momento. Esto se hace con la opción *Ejecutar-Algoritmo* de la última ventana.

Algoritmo. El algoritmo que se utiliza en este sistema se basa en el algoritmo Davis-Putnam [Davis-Putnam] y se describe en el capitulo 3 de esta tesis. Éste irá señalando cuando ejecuta cada una de las reglas descritas en dicho capítulo (R1 - R4), con el fin de dar un seguimiento la fórmula instanciada. Después de cada una de estas reglas hay que oprimir la tecla *ENTER*

En caso de que el algoritmo así lo indique, la fórmula será insatisfacible y su negación una verdad universal. En caso contrario, como en el ejemplo, habrá que segur

agregando instancias de cláusulas con el procedimiento mencionado hasta que estas se vuelvan inconsistentes aunque puede ser que esto nunca suceda.

En el ejemplo:

(4)	Qa/¬Pc;f(c)/¬Pc;g(c)	X5/c X6/c
(3)	Qa/Pf(c);c/¬Pc;g(c)	X3/c X4/g(c)

El conjunto de cláusulas instanciadas, se puede ver en la ventana: *Información-Instancias*

=f(c);c
Qa/Pc;g(c)
¬Qa
Qa/¬Pc;f(c)/¬Pc;g(c)
Qa/Pf(c);c/¬Pc;g(c)

Esta fórmula no tiene modelo.

Para terminar con el programa hay que teclear *Ctl-T*
Para regresar de cualquier pantalla anterior teclear "*ESC*".

Cualquier fórmula de la Lógica de Primer Orden y por lo tanto, toda sentencia de un universo posible, puede ser procesada con este método de demostración simbólica.

Otros ejemplos:

Fórmula	Forma Clausular
(a) $\forall x\exists y(Pxy \vee Pyx) \wedge \forall x\forall y(Pxy \rightarrow Pyy) \vdash \exists z(Pzz)$	Pxf(x) Pf(x)x ¬Pxy Pyy ¬Pzz
(b) $\forall x\forall y Pxy \vdash \forall y\exists x Pxy$	Pcy ¬Pxb
(c) $\exists x(Px \wedge \forall y(Qy \rightarrow Rxy)) \vdash \forall y(Qy \rightarrow \exists x(Px \wedge Rxy))$	Pc ¬Qy Rcy Qb ¬Px ¬Rzb
(c') $\exists x(Px \wedge \forall y(Qy \rightarrow Rxy)) \vdash \forall y(\exists x Px \wedge (Qy \rightarrow \exists z Rzy))$	Pc ¬Qy Rcy ¬Px Qb ¬Px ¬Rzb

(d) Sea φ:

A_1	$\forall x \forall y \forall z m(m(x,y),z) \approx m(x,m(y,z))$	Propiedad Asociativa
A_2	$\forall w m(w,e) \approx w$	Elemento Neutro p/derecha
A_3	$\forall u \exists x m(u,x) \approx e$	Elemrento Inverso p/derecha
T	$\forall x \forall y \forall z m(y,x) \approx m(z,x) \rightarrow y \approx z$	Ley de la Cancelación.

Forma Clausular de ¬φ:

m(m(x,y),z)≈m(x,m(y,z))
m(w,e) ≈w
m(u,f(u)) ≈e
m(b,a) ≈m(c,a)
b≠c

[Amor]
Amor J. A. 1988. Axiomatibilidad y Completud en Lógica de Primer Orden. Vínculos Matemáticos. No. 160. Ser. Notas de Clase. Facultad de Ciencias UNAM. México D.F.

[Amor Ramírez]
Amor J. A. Ramírez J. A. 1992. <<Un procedimiento de prueba automática en interacción con el usuario>>. Memoria IX Reunión Nacional de Inteligencia Artificial. SMIA 1992.

[Davis]
Davis Martin. A Computer Program for Presburguer's Algorithm. Summer Inst. For Symbolic Logic, Cornell U. V. 1957. Pp. 215-233.

[Davis - Putnam]
Davis Martin, Putnam Hillary <<A Computing Procedure for Quantification Theory.>> Journal of ACM, No. 3, 1960, pp.210-215.

[Davis 1983]
Davis Martin. The Prehistory and Early History of Automated Deduction. Automation of Reasoning, Classical Papers on Computational Logic 1957- 1966. Ser. Simbolic Computation No. 1,Alemania 1983, pp. 1-28.

[Gilmore]
Gilmore P. C. 1960. <<A Proof Method for Quantification Theory>>. Its Justification and Realization. IBM J.Res. Dev. 4. Pp. 28-35

[Lloyd]
Lloyd 1987. Foundations of Logic Programming. Springer Verlag.

[Loveland]
Loveland D. 1987. Automated Theorem Proving. North Holland.

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

[Murguía]
Miguel Murguía. Traductor de fórmulas de Lógica de Primer Orden a Forma Clausular Programa en Prolog. 1990

[Prawitz]
Prawitz D. 1960. Advances and Problems in Mechanical Proof Procedures. Univ of. Stockholm, 1969.

[Quaife]
Quaife Art Automated Development of Fundamental Mathematical Theories. Kluwer Academic Publishers. Dordrecht Holanda, 1992. Ser. Automated Reasoning, Vol. 2.

[Robinson A.]
Robinson A. <<Proving a Theorem (As Done by Man, Logician or Machine)>>. Summaries of Talks Presented at the Summer Institute for Symbolic Logic. Communications Res. Div. Inst. for Defense Analisys, Princeton, New Jersey, 1957, 2nd Edition 1960.

[Sabre]
Sabre S. 1988. Traducción Automática a Forma Clausular. Tesis para obtener el titulo de Matematico, UNAM.

[Sabre- Loyo]
Sabre S. Loyo C. 1989. Mejoras al algoritmo clásico de Skolemización. Aportaciones Matemáticas 6. S. M. M.

[Shostak]
Shostak R. 1978 <<An Algorithm for Reasoning about Equality>>. Comm. Of the ACM Vol. 21, No. 7.

[Siekmann - Wrightson]
Siekmann Jorg Wrightson graham. Classical Papers on Computational Logic 1957-1966. Ser. Automation of Reasoning. Springer Verlag 1983 Vol 1.

[Tarjan]
Tarjan R. 1975 <<Efficency of a Good but Non Linear Set Union Algorithm>>. JACM Vol. 22, No. 2

[Wang]
Wang H. <<Computer Theorem Proving and Artificial Intelligence>>. Contemporary Mathemalics Vol 29.