UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

DIVISIÓN DE ESTUDIOS DE POSGRADO

FUNDAMENTOS LÓGICOS DEL PROGRAMA DE RAZONAMIENTO AUTOMÁTICO OTTER

T E S I S

QUE PARA OBTENER EL GRADO ACADÉMICO DE

MAESTRO EN CIENCIAS
(MATEMÁTICAS)

P R E S E N T A

FAVIO EZEQUIEL MIRANDA PEREA

DIRECTOR DE TESIS: M. EN FIL. DE LA C. JOSÉ ALFREDO AMOR MONTAÑO

1999

Fundamentos Lógicos del Programa de Razonamiento Automático OTTER

Mat. Favio Ezequiel Miranda Perea

Abril de 1999

A Mis Padres, Ofelia y Pascual.

Por dejarme no sólo existir, sino ser plenamente.

A la memoria del Dr. Santiago Ramírez Castañeda.

Gracias Maestro.

A Carmen Gabriela.

Por el verde y eterno sentimiento que causas en mi.

A Liliana.

Por tus risas y por escucharme siempre.

A Elisa

Por "pervertirme" hacia las Ciencias de la Computación.

A José Alfredo y Carlos.

Por el apoyo continuo.

Al autodenominado grupo "La Palomilla".

Todavia no se porque, pero por algo será.

A Abigail, Armando, Isabel y Maria Elena.

Por soportarme y compartir la amistad durante 13 años.

Agradecimientos

Quiero agradecer a José Alfredo por presentarme a la nutria, aún queda mucho por descubrir y espero que sigamos en colaboración; a Atocha y Raymundo por su entusiasmo y dedicación al revisar este trabajo; a Carlos por la excelente corrección de estilo y a Raúl por las sugerencias que hicieron la exposición más accesible.

Un agradecimiento especial al Dr. Larry Wos, padre del razonamiento automático y miembro honorario de la división de Matemáticas y Ciencias de la Computación del Laboratorio Nacional de Argonne Ill. USA., por regalarme su libro [WOLB92], sin el cual habría sido imposible realizar este trabajo.
El regalo del Dr. Wos muestra que el verdadero gran científico es aquel que conserva la humildad.

A Eulogia y Ezequiel en dondequiera que estén...

Contenido

¿Qué es el Razonamiento Automático?

Para entender qué es un razonamiento automático primero debemos entender que es un razonamiento. *Razonamiento* es el proceso de obtener conclusiones a partir de hipótesis, a las que en adelante nos referiremos como hechos[1]. Para que el razonamiento sea correcto las conclusiones deben seguirse inevitablemente de los hechos. En esta situación nos referimos al *razonamiento lógico*, no razonamiento probabilistico o al sentido común. Las únicas conclusiones que son aceptables son las consecuencias lógicas de las hipótesis proporcionadas.
El objetivo del razonamiento automático es escribir progra[illegible]
que sirvan de a[illegible] la resolución de problema[illegible]
tas que requieran del razonamiento. La siguiente pregunta es ¿Como instruir a un programa de razonamiento automático para que lleve a cabo una tarea específica? El primer paso es "explicarle" al programa el problema que se quiere resolver, proporcionandole un conjunto de hechos que modelen la situación en cuestión, para lo cual se utilizan los lenguajes formales de la lógica matemática. En nuestro caso esoecífico utilizaremos la lógica de primer orden, aunque cabe señalar que las diversas lógicas de orden superior también sirven como base a programas de razonamiento automático, tema que no trataremos aquí.

Una vez que el programa conoce el problema, busca nuevos hechos generando conclusiones a partir de hechos anteriores, aplicando tipos específicos de razonamiento conocidos como *reglas de inferencia*. Los hechos obtenidos al aplicar las reglas de inferencia se agregan a la base de datos de información o conocimiento de acuerdo con ciertos criterios. La aplicación de reglas de inferencia a un conjunto de hechos de manera exhaustiva es una manera demasiado ingenua para atacar un problema. El intento de resolver aún el más simple de los problemas o acertijos mediante el ataque exhaustivo produciría una cantidad enorme de información, la mayor parte de ella irrelevante, por lo que es necesario controlar de una manera considerable la aplicación de reglas de inferencia.
Esta necesidad de control se cubre mediante una *estrategia* de razonamiento. En ajedrez, por ejemplo, jugar simplemente de acuerdo a las reglas sin evaluar las consecuencias generalmente lleva a perder el juego. En poker, apostar sola-

[1] Nos referimos únicamente al razonamiento deductivo.

mente de acuerdo a las posibilidades usualmente causa la pérdida del dinero. La estrategia es un aspecto importante de cada juego y es indispensable para ganar. Similarmente, un programa de razonamiento automático debe usar estrategias si quiere tener una oportunidad de resolver el problema que está atacando. Algunas estrategias guían a los programas de razonamiento en su elección de la información o conocimiento en la cual deben enfocarse. Es más, algunas estrategias hacen que el programa explore clases enteras de conclusiones. Incluso hay estrategias que permiten transmitir al programa de razonamiento automático la intuición y el conocimiento propios acerca de cómo resolver un problema. Podemos decir entonces que, un programa de razonamiento automático aplica reglas de razonamiento continuamente, pero sujetas a diversas estrategias.[2]

En este trabajo nos ocupamos de analizar algunos de los aspectos descritos anteriormente en el caso particular del programa de razonamiento automático OTTER, del laboratorio nacional de Argonne, Ill. USA, principalmente en lo que atañe a las reglas de inferencia que éste ofrece, así como a su estrategia principal.
Para que el lector obtenga una visión general de las aplicaciones de este programa sugerimos leer el artículo introductorio [Wos98] antes de este trabajo.

[2] Adaptación de la introducción a [WOLB92].

Capítulo 1

Preliminares

En lo que sigue, se supone un conocimiento medio de la sintaxis y la semántica de la lógica de primer orden con igualdad, incluyendo los teoremas más importantes. Aquí solo presentamos una introducción al lenguaje de cláusulas, así como el teorema de Herbrand y el teorema de Compacidad. Todo concepto no definido se supone conocido, pudiendose aclarar cualquier duda en [Mir99] o [SA91].

1.1 El lenguaje clausular

En esta sección presentamos las definiciones básicas del lenguaje de cláusulas, así como un algoritmo para transformar cualquier fórmula de un lenguaje de primer orden al lenguaje clausular.

Definición 1.1 Una *literal* es una fórmula atómica o su negación.

Las literales constituyen los objetos básicos del lenguaje clausular, a partir de los cuales se construyen las cláusulas.

Definición 1.2 Una *cláusula* es una disyunción de literales. Recursivamente:

- $\square$ es una cláusula, llamada la *cláusula* vacía.
- Si L es una literal entonces L es una cláusula.
- Si C, D son cláusulas entonces $C \vee D$ es una cláusula.
- Una expresión es una cláusula sólo si es de alguna de las formas descritas anteriormente.

Con $|C|$ denotamos al número de literales de la cláusula C. Además se observan las siguientes convenciones: $C \vee \square \vee D = C \vee D$ y $\square \vee \square = \square$.

Al preguntarnos si una fórmula o conjunto de fórmulas tiene o no un modelo a veces es difícil manipular las expresiones involucradas. Una manera de facilitar las cosas es obtener un conjunto lógicamente equivalente, aunque tal vez esto sea pedir demasiado. Aún así, podemos restringirnos a obtener un conjunto que comparta con el conjunto original la propiedad de tener un modelo. Esto motiva la siguiente definición.

Definición 1.3 Sean φ, ψ dos fórmulas. Decimos que φ, ψ son *satisfaciblemente equivalentes* cuando φ es satisfacible si y sólo si ψ es satisfacible. Esta propiedad sera denotada con $\varphi \sim_{sat} \psi$.

Es claro que $\varphi \equiv \psi$ implica que $\varphi \sim_{sat} \psi$, mas no al revés.

Ejemplo 1.1 Claramente $\forall x\ P(x,a) \sim_{sat} \forall x\ P(a,x)$ pero si interpretamos a como 0 y P como $\leq$ en $\mathbb{N}$ observamos que $\forall x\ P(x,a) \not\equiv \forall x\ P(a,x)$.

Ejemplo 1.2 $\exists x P(x) \sim_{sat} P(c)$. Este ejemplo muestra que una fórmula satisfaciblemente equivalente a la fórmula existencial dada, resulta mucho más fácil de manejar.

1.1.1 Formas Normales

En esta sección desarrollaremos diversas formas normales, que se obtienen transformando una fórmula sin perder su satisfacibilidad. Al terminar la sección podremos construir la forma clausular de cualquier fórmula dada.

Definición 1.4 Una fórmula φ está *rectificada* si y sólo si se cumplen las siguientes condiciones:

1. φ no tiene presencias libres y acotadas de una misma variable.
2. φ no tiene cuantificadores de la misma variable con alcances ajenos.
3. φ no tiene cuantificadores que acotan las mismas presencias de una variable (cuantificadores múltiples).
4. φ no tiene cuantificadores vacuos.

Siempre es posible transformar, de manera algorítmica, una fórmula φ en una fórmula $Rec(\varphi)$ de manera que $Rec(\varphi)$ esta rectificada y $Rec(\varphi) \equiv \varphi$.
La primera forma normal que presentamos es la llamada forma normal conjuntiva que consiste en expresar cualquier fórmula libre de cuantificadores como una conjunción de disyunciones de literales, es decir, como una conjunción de cláusulas.

Definición 1.5 Una fórmula φ está en *forma normal conjuntiva, (FNC)* si $\varphi = \bigwedge_{i \leq n} \chi_i$ y para cada i, $\chi_i = \bigvee_{j \leq m_i} L_{ij}$ donde para cada i, j, L_{ij} es una literal.

Proposición 1.1 *Para cualquier fórmula φ libre de cuantificadores se puede encontrar, de manera algorítmica, una fórmula ψ en forma normal conjuntiva tal que $\varphi \equiv \psi$. A ψ se le denota con $FNC(\varphi)$.*

dem:

Podemos suponer que φ solo contiene negaciones y conjunciones. Si φ es una literal L entonces ya está en FNC pues $L \equiv (L \vee L) \wedge (L \vee L)$. Análogamente si φ es una conjunción de literales entonces φ ya está en FNC, pues $\varphi \equiv \varphi \wedge \varphi$. En cualquier otro caso mediante distributividades o la siguiente equivalencia, obtenida con distributividad, se puede obtener la FNC deseada:
$(\alpha_1 \vee \ldots \vee \alpha_n) \wedge (\beta_1 \vee \ldots \vee \beta_m) \equiv (\alpha_1 \wedge \beta_1) \vee (\alpha_1 \wedge \beta_2) \vee \ldots (\alpha_1 \wedge \beta_m) \vee \ldots \vee (\alpha_n \wedge \beta_1) \vee (\alpha_n \wedge \beta_2) \vee \ldots (\alpha_n \wedge \beta_m)$. ⊣

Definición 1.6 Una fórmula de la forma $Q_1 x_1 \ldots Q_n x_n \chi$ donde χ es libre de cuantificadores y para toda i, $Q_i \in \{\forall, \exists\}$ se llama *forma normal prenex.*

Es fácil ver que para cualquier fórmula φ existe una fórmula ψ en forma normal prenex tal que $\varphi \equiv \psi$. Esta última fórmula se denota con $FNP(\varphi)$.
Ahora centramos nuestro interés en los cuantificadores existenciales y buscamos un posible método para eliminarlos de una fórmula, de manera que se preserve la satisfacibilidad. Para esto utilizaremos el proceso de *skolemización*[1], el cual consiste en eliminar los cuantificadores existenciales en favor de testigos de los mismos de la siguiente manera:

$\exists x \varphi$ se substituye por $\varphi\{x/c\}$

$\forall x_1 \ldots \forall x_n \exists y \varphi$ se substituye por $\forall x_1 \ldots \forall x_n \varphi\{y/g(x_1, \ldots, x_n)\}$

donde c es un nuevo símbolo de constante, llamado *constante de Skolem* y g es un nuevo símbolo de función llamado *función de Skolem.* Con $Sko(\varphi)$ denotamos cualquier skolemización de φ que se obtenga al eliminar, mediante este proceso, todos los cuantificadores existenciales de la fórmula φ de izquierda a derecha.

Ejemplo 1.3

φ	$sko(\varphi)$
$\exists y R(y)$	$R(c)$
$\forall x \exists y P(x,y)$	$\forall x P(x, g(x))$
$\forall x \forall z \exists y R(x, g(y), f(z))$	$\forall x \forall z R(x, g(h(x,z)), f(z))$

[1] El nombre es debido a su creador, Thoralf Skolem (1887-1963).

Si bien en general no es cierto que una fórmula sea lógicamente equivalente a su skolemización, con ayuda del siguiente teorema demostraremos que $\varphi \sim_{sat} sko(\varphi)$.

Lema 1.1 *Sean Σ una signatura, $\mathfrak{A}$ una Σ-interpretación, s un estado de las variables, t un término y φ una fórmula tal que $\varphi\{x/t\}$ es admisible. Si $a = t^{\mathfrak{A}}[s]$ entonces:*

$$\mathfrak{A} \models \varphi\{x/t\}[s] \Leftrightarrow \mathfrak{A} \models \varphi[s(x/a)]$$

dem:

Véase [Amo99, Mir99]. ⊣

Teorema 1.1 *(AE[2]) (Forma normal de Skolem).*
Sea $\forall x_1 \ldots \forall x_n \exists y \varphi$ una Σ-fórmula cerrada tal que las variables $x_1, \ldots, x_n$ no figuran acotadas en φ (así que $\varphi\{y/g(x_1, \ldots, x_n)\}$ es admisible), donde g es un nuevo símbolo de función n-ario. Sea $\Sigma_g = \Sigma \cup \{g\}$. Entonces todo Σ_g-modelo de $\forall x_1 \ldots \forall x_n \varphi\{y/g(x_1, \ldots, x_n)\}$ es un modelo de $\forall x_1 \ldots \forall x_n \exists y \varphi$. Inversamente, todo Σ-modelo de $\forall x_1 \ldots \forall x_n \exists y \varphi$ puede expandirse a un Σ_g-modelo de $\forall x_1 \ldots \forall x_n \varphi\{y/g(x_1, \ldots, x_n)\}$

dem:

La primera afirmación es inmediata a partir de la validez de la fórmula

$$\forall x_1 \ldots \forall x_n \varphi\{y/g(x_1, \ldots, x_n)\} \rightarrow \forall x_1 \ldots \forall x_n \exists y \varphi$$

Inversamente, sea $\mathfrak{A}$ un Σ-modelo de $\forall x_1 \ldots \forall x_n \exists y \varphi$, es decir, dado cualquier $(a_1, \ldots, a_n) \in A^n$, donde $|\mathfrak{A}| = A$, hay un $a \in A$ tal que si $s(x_i) = a_i$ entonces $\mathfrak{A} \models \varphi[s(y/a)]$.
Sea $G : A^n \longrightarrow A$ una función de elección tal que $G(a_1, \ldots, a_n) = a$ syss[3] $\mathfrak{A} \models \varphi[s(y/a)]$. Expandemos $\mathfrak{A}$ a un Σ_g-modelo $\mathfrak{B}$ tal que $g^{\mathfrak{B}} = G$. Por último, utilizando el lema 1.1 concluimos que para cada estado s se tiene que $\mathfrak{B} \models \varphi\{y/g(x_1, \ldots, x_n)\}$, así que $\mathfrak{B}$ es modelo de $\forall x_1 \ldots \forall x_n \varphi\{y/g(x_1, \ldots, x_n)\}$.
⊣

Corolario 1.1 *Si φ es una fórmula entonces $\varphi \sim_{sat} Sko(\varphi)$*

dem:

Es consecuencia inmediata del teorema. ⊣

A partir de todos los resultados anteriores podemos construir para cada fórmula su forma normal de Skolem.

[2] AE denota el uso del axioma de elección.
[3] De ahora en adelante "syss", abrevia la frase "si y sólo si".

Definición 1.7 Una fórmula cerrada, es decir sin variables libres, de la forma $\forall x_1 \ldots \forall x_n \varphi$, donde φ *no tiene cuantificadores y está en forma normal conjuntiva*, es una fórmula en *forma normal de Skolem.*

La condición de que φ esté en forma normal conjuntiva no es necesaria pero facilita las cosas.

Teorema 1.2 *Para cada fórmula φ podemos construir efectivamente una fórmula ψ en forma normal de Skolem de manera que $\varphi \sim_{sat} \psi$. Esta última fórmula se denota con $FNS(\varphi)$.*

dem:

Es consecuencia inmediata de todos los resultados anteriores de esta sección. ⊣

Por último obtenemos la forma clausular de una fórmula, que consiste simplemente en eliminar los cuantificadores universales en la forma normal de Skolem respectiva.

Definición 1.8 Sea φ una fórmula tal que $FNS(\varphi) = \forall x_1 \ldots \forall x_n \chi$. A χ se le llama la *forma clausular* de φ y se denota con $Cl(\varphi)$.

Observemos que $Cl(\varphi)$ es una conjunción de cláusulas libre de cuantificadores; además es inmediato que existe un proceso efectivo para construir $Cl(\varphi)$ y que $Cl(\varphi) \sim_{sat} \varphi$. Uno de nuestros intereses es probar la no satisfacibilidad de algunos conjuntos de fórmulas, por lo que habremos de trabajar con las respectivas formas clausulares que nos ofrecen diversas ventajas, como caracterizar ciertas propiedades semánticas mediante propiedades sintácticas.
Para un estudio a fondo acerca de procedimientos para obtener formas clausulares *véase* [Sab88].

1.1.2 El Paradigma

Existen dos maneras de cimentar la automatización del razonamiento. Una de ellas consiste en estudiar y emular a la gente, lo cual se conoce actualmente como *razonamiento orientado en personas* (véase [WV94]); la otra consiste en acudir a la lógica.
Entre los paradigmas para el razonamiento automático basados en lógica utilizaremos *el paradigma del lenguaje clausular.* Además de basarse en el uso de cláusulas para representar información, el paradigma del lenguaje clausular -en contraste con los paradigmas basados en la programación lógica- se basa en la retención de cláusulas, el uso de reglas de inferencia, así como en ciertas estrategias y otros procedimientos. De manera más específica podemos distinguir lo siguiente:

- Retención de claúsulas derivadas anteriormente.

- Uso de diversas reglas de inferencia para obtener conclusiones a partir de la descripción de un problema.
- Estrategias para restringir la obtención y retención de cláusulas, así como para controlar la aplicación de reglas de inferencia.
- Demodulación[4] para simplificar y estandarizar conclusiones.
- Subsunción[5] para eliminar una clase importante de conclusiones que son lógicamente más débiles que otras retenidas anteriormente.

El interés por implementar el paradigma del lenguaje clausular proviene de la necesidad de probar teoremas en diversas áreas de la matemática. Influenciados sin duda por la práctica constante en matemáticas de demostrar lemas que se usarán después en la demostración de un teorema, los investigadores han asumido tácitamente que la mejor manera de atacar un problema es reteniendo conclusiones derivadas durante el intento de completar una tarea dada.
Para una introducción avanzada al tema véase [WV94].

1.2 El Teorema de Herbrand

Las fórmulas en forma normal de Skolem son fórmulas cerradas y cuantificadas universalmente. El teorema de Herbrand muestra algunas de las ventajas de tales fórmulas y conjuntos de fórmulas.

Definición 1.9 Sea Σ una signatura con al menos una constante. Una *interpretación o estructura de Herbrand* es una Σ-interpretación $\mathfrak{A}$ tal que el universo de $\mathfrak{A}$ es el conjunto de términos cerrados de Σ (es decir, $|\mathfrak{A}| = \{t \mid t$ es un $\Sigma-$término cerrado$\}$), llamado el *universo de Herbrand* de Σ; la interpretación de cada símbolo de constante $c \in \Sigma$ es $c^{\mathfrak{A}} = c$ y la interpretación de los símbolos funcionales es:

$$f^{\mathfrak{A}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$$

Un *Modelo de Herbrand* para un conjunto de fórmulas M es una estructura de Herbrand que es un modelo de M.

Obsérvese que lo único que no está determinado en una estructura de Herbrand es la interpretación de los símbolos de predicado y que los símbolos de constante y de función se interpretan como ellos mismos. Por esta razón a los modelos de Herbrand también se les llama modelos sintácticos.

El siguiente lema dice cómo se evaluan los términos en una estructura de Herbrand:

[4] Este término se explicará más adelante.
[5] Idem.

Lema 1.2 *Sean Σ una signatura con al menos una constante y $\mathfrak{A}$ una Σ-interpretación de Herbrand. Sean t un término con variables $x_1, \ldots, x_n$ y s un estado de las variables tal que $s(x_i) = r_i$ donde r_i es un término cerrado, para $1 \leq i \leq n$. Entonces:*

$$t^{\mathfrak{A}}[s] = t\{x_1/r_1, \ldots, x_n/r_n\}$$

En particular si t es un término cerrado, $t^{\mathfrak{A}}[s] = t$

dem:

(Por inducción sobre los términos). Ejercicio. ⊣

El lema anterior dice que la interpretación de términos en una estructura de Herbrand consiste en aplicar una substitución al término, y que los términos cerrados se interpretan como ellos mismos.

Definición 1.10 Sean φ una fórmula y σ una sustitución. Decimos que $\varphi\sigma$ es una *instancia cerrada* de φ syss $\varphi\sigma$ no contiene variables libres.

Definición 1.11 Sean Σ una signatura con al menos una constante y M un conjunto de fórmulas universales cerradas. El conjunto de instancias cerradas $\varphi\sigma$ donde $\forall x_1 \ldots \forall x_n \varphi$ pertenece a M se llama *conjunto de instancias cerradas* de M y se denota $IC(M)$.

Obsérvese que si M consiste de fórmas normales de Skolem entonces

$$IC(M) = \{Cl(\varphi)\sigma \mid \varphi \in M \text{ y } Cl(\varphi)\sigma \text{ es una instancia cerrada}\}$$

es decir el conjunto de instancias cerradas de M es el conjunto de instancias cerradas de las formas clausulares de las fórmulas de M.

A continuación presentamos el Teorema Clásico de Herbrand en una forma moderna:

Teorema 1.3 *(Teorema de Herbrand).*
Sean Σ una signatura con al menos una constante y M un conjunto de Σ-enunciados universales. Las siguientes condiciones son equivalentes:

(a). M tiene un modelo.

(b). M tiene un modelo de Herbrand.

(c). $IC(M)$ tiene un modelo.

(d). $IC(M)$ tiene un modelo de Herbrand.

dem:

Debido a que todo modelo de Herbrand de M es un modelo de M, las implicaciones $b \Rightarrow a$ y $d \Rightarrow c$ son triviales.
La válidez universal de la fórmula $\forall x_1 \ldots \forall x_n \varphi \rightarrow \varphi\{x_1/t_1, \ldots, x_n/t_n\}$ hace inmediatas las implicaciones $a \Rightarrow c$ y $b \Rightarrow d$. Así que basta probar la implicación $c \Rightarrow b$.

Sea $\mathfrak{A}$ un modelo de $IC(M)$. Para definir una estructura de Herbrand $\mathfrak{B}$, basta decir cómo se interpretan los símbolos de predicado, pues por definición los símbolos de constante y de función se interpretan como ellos mismos. Si P es un símbolo de predicado n-ario, entonces definimos:

$$P^{\mathfrak{B}} = \{(t_1, \ldots, t_n) \mid \mathfrak{A} \models P(t_1, \ldots, t_n)\}$$

Obsérvese que, por construcción de $\mathfrak{B}$, se cumple que:

$$\mathfrak{A} \models P(t_1, \ldots, t_n) \Leftrightarrow \mathfrak{B} \models P(t_1, \ldots, t_n)$$

es decir, $\mathfrak{A}$ y $\mathfrak{B}$ validan a las mismas fórmulas atómicas cerradas. Dicho resultado se puede extender a cualquier fórmula cerrada libre de cuantificadores mediante inducción sobre fórmulas (ejercicio).
Por último veamos que $\mathfrak{B}$ es modelo de M. Sea $\forall x_1 \ldots \forall x_n \varphi$ una fórmula de M. Queremos demostrar que $\mathfrak{B} \models \forall x_1 \ldots \forall x_n \varphi$, lo cual es equivalente a mostrar que para cualesquiera $t_1, \ldots, t_n \in |\mathfrak{B}|$, $\mathfrak{B} \models \varphi\{x_1/t_1, \ldots, x_n/t_n\}$.[6] Pero esta última afirmación es equivalente, utilizando el último ejercicio, a lo siguiente:

$$\text{Para cualesquiera } t_1, \ldots, t_n \in |\mathfrak{B}|, \ \mathfrak{A} \models \varphi\{x_1/t_1, \ldots, x_n/t_n\}$$

lo cual es cierto pues $\varphi\{x_1/t_1, \ldots, x_n/t_n\}$ pertenece a $IC(M)$ y, por hipótesis, $\mathfrak{A}$ es modelo de $IC(M)$. De manera que $\mathfrak{B} \models \forall x_1 \ldots \forall x_n \varphi$ y $\mathfrak{B}$ es un modelo de Herbrand de M. ⊣

1.3 El Teorema de Compacidad

En esta sección enunciamos el teorema de Compacidad, el cual es crucial en la automatización del razonamiento, ya que permite reducir el problema de la satisfacibilidad de un conjunto arbitrario de fórmulas al de la satisfacibilidad de sus subconjuntos finitos.
Enunciamos el teorema para el caso de fórmulas cerradas y libres de cuantificadores, es decir, fórmulas sin variables.

Teorema 1.4 *(AE) (Compacidad para conjuntos de fórmulas cerradas y libres de cuantificadores).*
Sean Σ una signatura y M un conjunto de Σ-fórmulas cerradas y libres de cuantificadores. Entonces, M tiene un modelo si y sólo si todo subconjunto finito de M tiene un modelo.

[6] Recuérdese que $|\mathfrak{B}| = \{t \mid t \text{ es un } \Sigma - \text{término cerrado}\}$

dem:

La demostración utiliza el lema de König dependiente del axioma de elección y puede verse en [SA91]. Otra demostración clásica, que sigue el método de Henkin y también depende del axioma de elección puede verse en [Amo99]. ⊣

Ahora, a partir de los teoremas de Herbrand y de Compacidad restringido, demostramos el teorema general de Compacidad.

Teorema 1.5 *(AE) (Teorema de Compacidad).*
Sean Σ una signatura y M un conjunto de Σ-fórmulas cerradas. Entonces M tiene un modelo si y sólo si todo subconjunto finito de M tiene un modelo.

dem:

$\Rightarrow$) Es trivial.
$\Leftarrow$) Supongamos que M no tiene un modelo.
Sea $M' = FNS(M)$, el conjunto de formas normales de Skolem de los elementos de M, donde para cada $\varphi \in M$ se usó un juego diferente de funciones de Skolem. Puesto que $\varphi \sim_{sat} FNS(\varphi)$, se tiene que M' no tiene modelo. Por el teorema de Herbrand, concluimos que $IC(M')$ no tiene modelo y como $IC(M')$ consta de fórmulas cerradas y libres de cuantificadores, utilizando el teorema anterior, concluimos que existe un subconjunto finito $E' \subseteq M'$ tal que E' no tiene modelo. Pero tal E' tiene que ser de la forma $E' = FNS(E)$, donde E es algún subconjunto finito de M, de manera que E no puede tener modelo. ⊣

Corolario 1.2 *Sean Σ una signatura y M un conjunto de Σ-fórmulas. Entonces, M es satisfacible si y sólo si todo subconjunto finito de M es satisfacible.*

dem:

Utilizar que para cualquier fórmula φ se tiene que $\varphi \sim_{sat} \exists\varphi$. ⊣

El siguiente corolario, proporciona otra forma del teorema de Herbrand.

Corolario 1.3 *Sean Σ una signatura con al menos una constante y M un conjunto de Σ-fórmulas cerradas y universales. Sea $\exists x\psi$ una fórmula cerrada, con ψ libre de cuantificadores. Las siguientes condiciones son equivalentes:*

1. $M \models \exists x\psi$
2. *Existe un número finito de términos cerrados, $t_1, \ldots, t_n$ tales que* $M \models \psi\{x/t_1\} \vee \ldots \vee \psi\{x/t_n\}$

dem:

Véase [Mir99, SA91]. ⊣

Para la descripción de procedimientos de demostración automática que se basan en esta versión del teorema de Herbrand véase [AM98, Ram96].

1.4 Unificación

Los sistemas de programación lógica tienen dos puntos de partida, la regla de resolución que analizaremos en el siguiente capítulo y el proceso de unificación, que consiste en encontrar, dado un conjunto de literales o términos W, una substitución σ de tal forma que el conjunto imagen $W\sigma$ conste de un solo elemento. Como lo veremos más adelante, además de sus aplicaciones en programación lógica, la unificación también es importante para los sistemas de reescritura de términos y para el razonamiento automático, al ser la herramienta principal para el funcionamiento correcto de la regla de resolución. Para un panorama general del tema sugerimos consultar [BS94, Llo87]

Definición 1.12 Una *expresión* es una literal o un término.

Definición 1.13 Sea W un conjunto no vacío de expresiones. Un *unificador* de W es una substitución σ tal que $\mid W\sigma \mid = 1$.

Ejemplo 1.4 Sea $W = \{P(x, f(y)), P(x, f(x)), P(u, v)\}$ entonces la sustitución $\sigma = \{x/a, y/a, u/a, v/f(a)\}$ es un unificador de W, ya que $W\sigma = \{P(a, f(a))\}$

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

Definición 1.14 Un unificador σ de un conjunto de fórmulas W, se llama *unificador más general* (abreviado *umg*) si para cada unificador τ de W, existe una sustitución ϑ, tal que $\sigma\vartheta = \tau$.

Ejemplo 1.5 Sean $W = \{f(g(a, x), g(y, b)), f(z, g(u, v))\}$, $\tau = \{x/a, z/g(a, a)$ $y/u, v/b\}, \sigma = \{z/g(a, x), y/u, v/b\}$. Entonces τ y σ son unificadores de W y σ resulta ser umg; en particular si $\vartheta = \{x/a\}$ entonces $\sigma\vartheta = \tau$.

1.4.1 El Algoritmo de Unificación

Definición 1.15 Sea $W \neq \varnothing$ un conjunto de expresiones. Definimos el *conjunto discorde de* W como sigue:
Se localiza la posición del símbolo más a la izquierda en el que al menos dos expresiones de W difieren y se extrae de cada expresión de W la subexpresión que inicia en esta posición; el conjunto D_W de dichas subexpresiones es el *conjunto discorde de* W.

Ejemplo 1.6 Si $W = \{f(x, y), g(a)\}$ entonces $D_W = W$.

Ejemplo 1.7 Si $W = \{P(x, f(y)), P(x, f(a)), P(x, f(h(y)))\}$ entonces $D_W = \{y, a, h(y)\}$.

A continuación presentamos el algoritmo de unificación AU:

Precondición: $k = 0$, $\sigma_0 = \varnothing$
  while $W\sigma_k$ no sea unitario **do**
    Hallar el conjunto discorde D_k de $W\sigma_k$;
    if existen $v, t \in D_k$ tales que v no figura en t **then**
      $\sigma_{k+1} := \sigma_k\{v/t\}$;
      $k := k + 1$
    else
      write('W no es unificable');
      halt
    end if
  end while
Postcondición: σ_k es umg de W ó W no es unificable.

Para finalizar la sección haremos el análisis del algoritmo de unificación mostrando que la especificación:

$$\{k = 0,\ \sigma_0 = \varnothing\}\ AU\ \{\sigma_k \text{ es umg de } W \text{ o } W \text{ no es unificable }\}$$

es correcta.

Lema 1.3 *(Terminación de AU)*
Si m es el número de variables que figuran en W, entonces el máximo k tal que σ_k es calculado en AU es menor o igual que m.

dem:

Obsérvese que todas las variables que figuran en $\sigma_k, W\sigma_k, D_k$ son variables de W, de manera que AU no introduce variables nuevas.
Al calcular $\sigma_{k+1} := \sigma_k\{v/t\}$, la variable v que figuraba en W se sustituye por un término que no la contiene, por lo que al calcular $W\sigma_k$ se reduce el número de variables en uno y como hay m variables, este proceso se puede efectuar a lo más m veces. ⊣

Lema 1.4 *(Correctez parcial para un conjunto no unificable)*
Si W no es unificable, AU termina con el mensaje 'W no es unificable'.

dem:

Como $W\sigma_k$ no es nunca un unitario y AU siempre termina, por el lema anterior, necesariamente debe terminar con el mensaje requerido. ⊣

Lema 1.5 *(Las sustituciones intermedias son más generales)*
Sea ϑ un unificador de W. Si σ_k es la sustitución obtenida en la k-ésima iteración de AU, entonces existe una sustitución τ_k tal que $\vartheta = \sigma_k\tau_k$.

dem:

Inducción sobre k.

- $k = 0$ es obvio, al tomar $\tau_0 = \vartheta$.
- Hipótesis de Inducción: Existe un τ_k tal que $\vartheta = \sigma_k\tau_k$.
- Paso inductivo. Basta analizar el caso en que $W\sigma_k$ no es un unitario, pues si lo fuera el algoritmo habría terminado. Como $W\sigma_k$ no es unitario, AU produce el conjunto discorde D_k de $W\sigma_k$. Por HI, tenemos que $\vartheta = \sigma_k\tau_k$ y como ϑ unifica a W, se sigue que τ_k unifica a D_k. Por lo tanto debe existir una variable $v \in D_k$. Sea t cualquier otro término de D_k. Sabemos que v no figura en t, pues $v\tau_k = t\tau_k$. Sea $\sigma_{k+1} = \sigma_k\{v/t\}$. Definimos $\tau_{k+1} = \tau_k \setminus \{v/v\tau_k\}$. Debemos a considerar 2 casos:

  1. $v\tau_k \neq v$. Entonces

$$\begin{aligned} \tau_k &= \{v/v\tau_k\} \cup \tau_{k+1} \\ &\quad (v\tau_k = t\tau_k) \\ &= \{v/t\tau_k\} \cup \tau_{k+1} \\ &\quad (t\tau_k = t\tau_{k+1}) \\ &= \{v/t\}\tau_{k+1} \end{aligned}$$

  2. $v\tau_k = v$. En este caso se tiene que D_k sólo tiene variables como elementos, puesto que τ_k unifica a D_k; además $\tau_{k+1} = \tau_k$, por lo que $\tau_k = \{v/t\}\tau_{k+1}$.

  Así que en cualquier caso, $\tau_k = \{v/t\}\tau_{k+1}$, de donde se sigue que $\vartheta = \sigma_k\tau_k = \sigma_k\{v/t\}\tau_{k+1} = \sigma_{k+1}\tau_{k+1}$.

⊣

Lema 1.6 *(Correctez Parcial para un conjunto unificable)*
Si W es unificable entonces la ultima sustitución generada por AU es un umg de W.

dem:

Sean ϑ un unificador de W y σ_m la última sustitución calculada por AU. Por el lema 1.5 sabemos que existe τ_m tal que $\vartheta = \sigma_m\tau_m$. Como $W\vartheta$ es unitario, se sigue que τ_m unifica a $W\sigma_m$, es decir, se cumple la condición del ciclo *while*; pero como $\vartheta = \sigma_m\tau_m$ y ϑ era arbitrario, concluimos que σ_k es umg de W. ⊣

Teorema 1.6 *(Correctud total de AU)*
Dado un conjunto finito de expresiones W, el algoritmo AU termina dando como resultado el mensaje 'W no es unificable', en el caso en que W no es unificable, y un umg de W en el caso en que W es unificable.

dem:

Es consecuencia inmediata de los lemas 1.3, 1.4, 1.5, 1.6. ⊣

1.5 El Método de Davis-Putnam

El primer demostrador automático de teoremas para la lógica de primer orden fue escrito por Gilmore en 1960 [Gil60]; estaba esencialmente basado en el teorema de Herbrand y reducía el problema de no satisfacibilidad a la no satisfacibilidad proposicional. Tal método también puede consultarse en [Lei97].
Dicha implementación no pudo obtener pruebas de algunas fórmulas simples de la lógica de predicados. Una posible mejora en el método consistió en evitar la transformación a forma normal disyuntiva, desarrollando métodos de decisión para satisfacibilidad en forma normal conjuntiva. Una mejora de este tipo fue lograda poco después por Davis y Putnam [DP60].
Al igual que el método de Gilmore, el método de Davis-Putnam (DP) se basa en la construcción sucesiva de conjuntos de cláusulas cerradas, de los cuales se prueba su satisfacibilidad. Aunque no es adecuado para la lógica clausal de predicados, es eficiente para probar la satisfacibilidad de formas normales conjuntivas proposicionales; además, lo usaremos más adelante como una herramienta para algunas demostraciones.

Definición 1.16 Una cláusula C se llama *reducida* si cada literal en ella figura a lo más una vez.

Definición 1.17 Si L es una literal, definimos la *literal contraria* de L, denotada L^c, como sigue:

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

donde P es un predicado. Las literales L y L^c son *complementarias*.

A continuación definimos las cuatro reglas del método de Davis-Putnam. Sea $\mathbb{S}$ un conjunto de cláusulas cerradas y reducidas.

1. Regla de la Tautología (RT). Elimine de $\mathbb{S}$ todas las cláusulas que contienen literales complementarias, es decir, las tautologías.

2. Regla de la Cláusula Unitaria (RCU). Si $C \in \mathbb{S}$ y C es una cláusula unitaria, entonces elimine de $\mathbb{S}$ todas las cláusulas que contengan a C y elimine C^c de las cláusulas restantes.

3. Regla de la Literal Pura (RLP). Si $\mathbb{D} \subseteq \mathbb{S}$ es un conjunto tal que existe una literal L que figura en todas las cláusulas de $\mathbb{D}$ y L^c no figura en $\mathbb{S}$, entonces reemplace $\mathbb{S}$ por $\mathbb{S} - \mathbb{D}$.

4. Regla de Descomposición (RD). Si existe una literal L tal que $\mathbb{S}$ se puede descomponer como $\mathbb{S} = \{A_1, \ldots, A_n, B_1, \ldots, B_m\} \cup R$ y se cumplen las siguiente condiciones:

   (a) En R no figura ni L ni L^c.

   (b) Todas las A_i contienen a L pero no a L^c.

(c) Todas las B_j contienen a L^c pero no a L.

entonces reemplace $\mathbb{S}$ por los siguientes conjuntos

$$\mathbb{S}_1 = \{A'_1, \ldots, A'_n\} \cup R$$
$$\mathbb{S}_2 = \{B'_1, \ldots, B'_m\} \cup R$$

donde $A'_i = A_i \setminus L$ y $B'_j = B_j \setminus L^c$.[7]

Definición 1.18 Sea $\mathbb{S}$ un conjunto de cláusulas cerradas y reducidas. Con $\mathbb{S}^{\sim}$ denotamos al conjunto que resulta al aplicar a $\mathbb{S}$ alguna de las reglas RT, RCU, RLP. Con $\mathbb{S}_1, \mathbb{S}_2$ denotamos a los conjuntos que resultan al aplicar a $\mathbb{S}$ la regla RD.

El siguiente teorema asegura que las reglas anteriores son correctas

Teorema 1.7 *(Las reglas de DP son correctas). Sea $\mathbb{S}$ un conjunto de cláusulas cerradas y reducidas. Entonces:*

1. *Si alguna de las reglas RT,RCU,RLP es aplicable, $\mathbb{S} \sim_{sat} \mathbb{S}^{\sim}$.*
2. *Si la regla RD es aplicable, entonces, $\mathbb{S}$ es satisfacible si y sólo si $\mathbb{S}_1$ es satisfacible o $\mathbb{S}_2$ es satisfacible.*

dem:

la demostración es rutinaria y se puede consultar en [Lei97]. ⊣

Definición 1.19 Un *árbol de Davis-Putnam* o *DP-árbol* $\mathcal{T}$ para un conjunto finito de clásulas cerradas reducidas $\mathbb{S}$ es un árbol con las siguientes propiedades:

1. Los nodos de $\mathcal{T}$ son conjuntos finitos de cláusulas reducidas.
2. Los nodos de $\mathcal{T}$ son de grado menor o igual que 2, puesto que las reglas de DP, reemplazan un conjunto a lo más con dos conjuntos.
3. La raíz es $\mathbb{S}$.
4. Si $\mathbb{S}'$ es un nodo de grado 1 y $(\mathbb{S}', \mathbb{S}'')$ es un arco en $\mathcal{T}$, entonces $\mathbb{S}'' = \mathbb{S}'^{\sim}$ para alguna aplicación de RT,RCU o RLP.
5. Si $\mathbb{S}'$ es un nodo de grado 2 y $(\mathbb{S}', \mathbb{M})$, $(\mathbb{S}', \mathbb{N})$ son arcos en $\mathcal{T}$ (en ese orden) entonces $\mathbb{M} = \mathbb{S}'_1$ y $\mathbb{N} = \mathbb{S}'_2$ para alguna aplicación de RD.

Un DP-árbol para $\mathbb{S}$ es un *DP-árbol de decisión* para $\mathbb{S}$ si todas sus hojas son $\square$ o alguna hoja es $\varnothing$.

[7] La notación $C \setminus L$ indica que se elimine la literal L de la cláusula C.

Ejemplo 1.8 Sea $\mathbb{D} = \{P \vee Q, R \vee S, \neg R \vee S, R \vee \neg S, \neg R \vee \neg S, P \vee \neg Q \vee \neg P\}$
Aplicando RT a $\mathbb{D}$ obtenemos $\mathbb{D}_1 = \mathbb{D} - \{P \vee \neg Q \vee \neg P\}$.
Aplicando RLP a $\mathbb{D}_1$ obtenemos $\mathbb{D}_2 = \{R \vee S, \neg R \vee S, R \vee \neg S, \neg R \vee \neg S\}$.
Aplicando RD a $\mathbb{D}_2$ obtenemos $\mathbb{D}_{21} = \{R, \neg R\} = \mathbb{D}_{22}$.
Aplicando RCU a $\mathbb{D}_{21}, \mathbb{D}_{22}$ obtenemos $\{\square\}$.
De manera que el árbol $\mathcal{T}$ cuyos conjuntos de nodos y arcos son

$$Nod(\mathcal{T}) = \{\mathbb{D}, \mathbb{D}_1, \mathbb{D}_2, \mathbb{D}_{21}, \mathbb{D}_{22}, \{\square\}\}$$

$$Ar(\mathcal{T}) = \{(\mathbb{D}, \mathbb{D}_1), (\mathbb{D}_1, \mathbb{D}_2), (\mathbb{D}_2, \mathbb{D}_{21}), (\mathbb{D}_2, \mathbb{D}_{22})(\mathbb{D}_{21}, \{\square\}), (\mathbb{D}_{22}, \{\square\})\}$$

es un DP-árbol de decisión para $\mathbb{D}$.

Definición 1.20 Sea $\mathcal{T}$ un DP-árbol de decisión para $\mathbb{S}$. Decimos que $\mathcal{T}$ prueba la satisfacibilidad de $\mathbb{S}$ si hay una hoja que es $\varnothing$, y decimos que $\mathcal{T}$ prueba la no satisfacibilidad de $\mathbb{S}$ si todas las hojas son $\square$.

Resulta evidente a partir de la definición anterior que encontrar un DP-árbol de decisión para $\mathbb{S}$ de manera algorítmica decide su satisfacibilidad. Para facilitar la prueba de los teoremas de correctud y completud del método supondremos s.p.g. que $\mathbb{S}$ *no contiene tautologías*. Obsérvese que RCU, RLP y RD no introducen tautologías, de manera que la eliminación de las tautologías puede considerarse como un paso previo. Decimos que un conjunto de cláusulas está *taut-reducido* si no contiene tautologías.

Ahora ya podemos formular los teoremas de correctud y completud para el método de Davis-Putnam.

Teorema 1.8 *(Correctud de DP). Sea $\mathbb{S}$ un conjunto de cláusulas cerradas reducidas y taut-reducidas y sea $\mathcal{T}$ un DP-árbol de decisión para $\mathbb{S}$. Si $\mathcal{T}$ prueba la satisfacibilidad (no satisfacibilidad) de $\mathbb{S}$ entonces $\mathbb{S}$ es en realidad satisfacible (no satisfacible).*

dem:

Resulta inmediato mediante inducción sobre el número de nodos de $\mathcal{T}$, utilizando el hecho de que las reglas de DP son correctas. ⊣

Teorema 1.9 *(Completud de DP).*
Si $\mathbb{S}$ es un conjunto finito de cláusulas cerradas,reducidas y taut-reducidas y si $\mathcal{T}$ es un DP-árbol para $\mathbb{S}$, entonces $\mathcal{T}$ puede extenderse a un DP-árbol de decisión $\mathcal{T}'$.

dem:

Por inducción sobre el número de átomos que figuran en $\mathbb{S}$, denotado con $at(\mathbb{S})$.

- Base de la inducción. $at(\mathbb{S}) = 0$. En tal caso tenemos que $\mathbb{S} = \varnothing$ o $\mathbb{S} = \{\square\}$ y en ambos casos tenemos que la raíz $\mathbb{S}$ forma un DP-árbol de decisión.

- HI. Para $at(\mathbb{S}) \leq n$
- Paso Inductivo. $at(\mathbb{S}) = n + 1$. Vamos a distinguir dos casos:

  1. $\mathcal{T}$ consiste solamente de la raíz $\mathbb{S}$. Como $at(\mathbb{S}) \neq 0$ entonces debe haber una cláusula en $\mathbb{S}$ que contiene una literal L. Si L^c no figura en $\mathbb{S}$ entonces RLP es aplicable. Si L figura como cláusula unitaria entonces RCU es aplicable. Si L no figura ni como cláusula unitaria ni como literal pura entonces RD es aplicable. Así, alguna de las reglas RCU, RLP, RD es aplicable y, de la definición de las reglas, se sigue que $at(\mathbb{S}^{\sim}) \leq n$ o $at(\mathbb{S}_1), at(\mathbb{S}_2) \leq n$.
  Considerense las siguientes extensiones de $\mathcal{T}$ a $\mathcal{T}'$:

  $$Nod(\mathcal{T}') = \{\mathbb{S}, \mathbb{S}^{\sim}\}, Ar(\mathcal{T}') = \{(\mathbb{S}, \mathbb{S}^{\sim})\} \text{ o}$$
  $$Nod(\mathcal{T}') = \{\mathbb{S}, \mathbb{S}_1, \mathbb{S}_2\}, Ar(\mathcal{T}') = \{(\mathbb{S}, \mathbb{S}_1), (\mathbb{S}, \mathbb{S}_2)\}.$$

  Por HI cualquier DP-árbol con raíz $\mathbb{S}^{\sim}$ ($\mathbb{S}_1, \mathbb{S}_2$ respectivamente) puede extenderse a un DP-árbol de decisión. En particular, $\mathcal{T}$ puede extenderse a un DP-árbol de decisión, agregando a $\mathbb{S}$ como nueva raíz del árbol dado por la HI.

  2. $\mathcal{T}$ contiene arcos.
  Sean $\mathbb{D}_1, \ldots, \mathbb{D}_k$ todas las hojas de $\mathcal{T}$. Como las $\mathbb{D}_i$ son hojas, entonces $at(\mathbb{D}_i) \leq n$. Sean $\mathcal{T}_i$ los árboles que constan únicamente de la raíz $\mathbb{D}_i$. Por HI, cada $\mathcal{T}_i$ puede extenderse a un DP-árbol de decisión $\mathcal{T}_i'$. Sea $\mathcal{T}'$ el árbol obtenido sustituyendo en $\mathcal{T}$ las hojas $\mathbb{D}_i$ por los árboles $\mathcal{T}_i'$. Por construcción $\mathcal{T}'$ es una extensión de $\mathcal{T}$. Además como las hojas de $\mathcal{T}'$ son las hojas de los $\mathcal{T}_i'$, entonces $\mathcal{T}'$ es un DP-árbol de decisión.

⊣

En realidad el teorema anterior es un resultado de completud fuerte que muestra que sin importar cómo empecemos a aplicar las reglas de DP, siempre obtendremos un DP-árbol de decisión. Esto significa que cualquier DP-árbol que no se pueda extender propiamente es en realidad un DP-árbol de decisión.

En este capítulo hemos presentado las herramientas necesarias para el estudio de la regla de resolución así como una introducción al paradigma del lenguaje clausular, el cual es utilizado por OTTER.
El resultado de mayor relevancia derivado de este capítulo consiste en la reducción del problema de busqueda de un modelo para un conjunto de fórmulas a la búsqueda de un modelo para los subconjuntos finitos del conjunto de instancias cerradas de las formas cláusulares de las fórmulas del conjunto original. Sin embargo, tal reducción está muy lejos de ser práctica, aún cuando tenemos métodos de decisión como el de Davis-Putnam.
En el siguiente capítulo estudiamos la regla de resolución binaria, que nos ayudará a optimizar la solución del problema; también estudiaremos los refinamientos de resolución, especialmente aquellos que generan procesos implementados en OTTER.

Capítulo 2

Resolución y sus Refinamientos

2.1 Definiciones preliminares

Nuestro proposito es desarrollar el principio de resolución con unificación, presentando su completud como un procedimiento de refutación. Empezaremos con algunas definiciones:

Definición 2.1 Una substitución σ es una *permutación* si σ es inyectiva y $ran\ \sigma \subset VAR$, es decir, el rango de σ consta de variables. Una permutación es un *renombre* de un conjunto de expresiones E, si $V(E) \cap ran\ \sigma = \varnothing$.[1]

Definición 2.2 Una cláusula C es una *variante* de una cláusula D , si existe una permutación σ tal que $C\sigma = D$.

Definición 2.3 Sean C, D cláusulas y C_1, D_1 variantes respectivas tales que $V(C_1) \cap V(D_1) = \varnothing$. Supongamos que $C_1 : A \vee L \vee B$, $D_1 : E \vee M \vee F$ de tal forma que $\{L^c, M\}$ es unificable mediante el umg σ.
Entonces la cláusula $(A \vee B \vee E \vee F)\sigma$ es un *resolvente binario* de C y D. El átomo de L es el átomo sobre el cual se resolvió.

Definición 2.4 Sea C una cláusula y $\mathcal{A}$ un conjunto no vacío de literales que figuran en C. Si $\mathcal{A}$ es unificable mediante el umg μ entonces $C\mu$ se llama una *instancia general o G-instancia* de C.

Definición 2.5 Sea C una cláusula. Si C' se obtiene de C omitiendo algunas literales múltiples, se dice que C' es un *reducto* de C.

Definición 2.6 Un reducto de una G-instancia de una cláusula C es un *factor* de C. Un factor C' es no trivial si el número de literales de C' es estrictamente menor que el número de literales de C.

[1] $V(E)$ es el conjunto de variables de E.

Obsérvese que los conceptos de reducto y factor coinciden en cláusulas cerradas y que todo reducto es un factor trivial.

Definición 2.7 Sean C, D cláusulas con factores C', D' respectivamente. Si E es un resolvente binario de C', D' decimos que E es un *resolvente* de C y D.

Por último definimos formalmente lo que es una R-derivación o deducción:

Definición 2.8 Sea $\mathbb{S}$ un conjunto de cláusulas y C una cláusula. Una sucesión de cláusulas $C_1, \ldots, C_n$ es una *R-derivación* de C a partir de $\mathbb{S}$ si se cumple lo siguiente:

1. $C_n = C$
2. Para toda $i = 1, \ldots, n$ sucede alguna de las siguientes condiciones:
   (a) C_i es variante de alguna cláusula de $\mathbb{S}$
   (b) C_i es resolvente de C_j, C_k con $j, k < i$

Notación: $\mathbb{S} \underset{R}{\vdash} C$ significa que existe una R-deducción de C a partir de $\mathbb{S}$. Si $\mathbb{S} \underset{R}{\vdash} \square$ decimos que se obtiene una R-refutación de $\mathbb{S}$.

2.2 Completud de la Resolución

La prueba de completud refutacional de la resolución consiste esencialmente en la técnica de "levantamiento". Levantamiento es el reemplazo de una R-derivación por una R-derivación más general. Esto significa que cada instanciación puede ser reemplazada por un umg.

Definición 2.9 Sean $\Gamma : C_1, \ldots, C_n$, $\Delta : D_1, \ldots, D_n$ R-derivaciones. Decimos que Γ es *más general que* Δ si existen sustituciones $\vartheta_1, \ldots, \vartheta_n$ tales que $C_i\vartheta_i = D_i$, esto lo denotamos con $\Gamma \leq_s \Delta$. En particular si C, D son cláusulas definimos $C \leq_s D$ si existe una sustitución σ tal que $C\sigma = D$.

Los siguientes dos lemas muestran que la propiedad de levantamiento se cumple para la factorización y la resolución.

Definición 2.10 Si X' es un factor de una cláusula X, se define el *grado* de X' como el número de literales que permanecen en X al eliminar X', lo cual se denota con $|X| - |X'|$. Decimos que X' es un k-factor de X syss $|X| - |X'| = k$

Teorema 2.1 *(Los factores de instancias son instancias de factores).*
Sean C, D cláusulas tales que $C \leq_s D$.
Si F es un factor de D entonces existe un factor E de C tal que $E \leq_s F$.

dem:

Inducción sobre el grado del factor F.

- Base de la Inducción. Supongamos que F es un factor trivial.
  En este caso tenemos, por definición, que $F = D\sigma$ donde $D\sigma$ es una G-instancia de D. Además por hipótesis hay una sustitución λ tal que $D = C\lambda$. Por lo tanto, si tomamos $E = C$, tenemos que E es un factor (trivial) de C y $E\lambda\sigma = F$, es decir, $E \leq_s F$.

- Hipótesis de Inducción: Si F *es un* k*-factor de* D *entonces existe un factor* E de C tal que $E \leq_s F$.

- Paso Inductivo: Supongamos que F es un (k+1)-factor de D.
  Por definición tenemos que $|D| - |F| = k + 1$. Ahora bien, como F es un reducto de una G-instancia de D y al eliminar F de D, en D quedan k+1 literales, entonces podemos eliminar primero k de esas k+1 literales en D obteniendo así un k-factor F' de D tal que F es un 1-factor de F'. Es más F es un reducto de F'.
  Así F' debe ser de la forma $F' = F_1 \vee L \vee F_2 \vee L \vee F_3$ y s.p.g. $F = F_1 \vee L \vee F_2 \vee F_3$.
  Aplicando la hipótesis de inducción al k-factor F' de D existe un factor E' de C tal que $E' \leq_s F'$, digamos que $E' = E_1 \vee M_1 \vee E_2 \vee M_2 \vee E_3$, donde hay una una λ tal que

$$E_1\lambda = F_1, \quad E_2\lambda = F_2, \quad E_3\lambda = F_3, \quad M_1\lambda = M_2\lambda = L$$

  observemos dos cosas:

  1. M_1, M_2 pueden ser literales diferentes, pero λ unifica a $\{M_1, M_2\}$.
  2. E' es un reducto de una G-instancia de C, digamos que es reducto de $C\tau$.

  Debemos analizar dos casos:

  - $M_1 = M_2$. En este caso definimos $E = E_1 \vee M_1 \vee E_2 \vee E_3$, así claramente E es un reducto de E' y como E' es factor de C (por la hipótesis de inducción), entonces E es un factor de C, tal que $E\lambda = F, \therefore\ E \leq_s F$.
  - $M_1 \neq M_2$. En este caso en C figuran dos literales K_1, K_2 tales que $K_1\tau = M_1, K_2\tau = M_2$ (por la observación 2). Obviamente τ no unifica a $\{M_1, M_2\}$, pero como λ sí unifica a este conjunto entonces existe un umg σ de $\{M_1, M_2\}$ tal que $\sigma \leq_s \lambda$. Sea ϑ tal que $\sigma\vartheta = \lambda$. Por definición sabemos que $C\tau\sigma$ es una G-instancia de C. Es más existe una literal L_0 tal que $M_1\sigma = M_2\sigma = L_0$ y $L_0\vartheta = L$.
    Sean $E'_i = E_i\sigma$ para $i = 1, 2, 3$ y tomemos $E'' = E'_1 \vee L_0 \vee E'_2 \vee L_0 \vee E'_3$. De esta manera E'' es un reducto de $C\tau\sigma$, así que E'' es un factor de C y como $E' \leq_s F'$ (i.e., $E'\lambda = F'$) y σ es umg de $\{M_1, M_2\}$, entonces $E'' \leq_s F'$ (es fácil cerciorarse de que $E''\vartheta = F'$ [2]).
    Por último hagamos

$$E = E'_1 \vee L_0 \vee E'_2 \vee E'_3$$

[2] $E''\vartheta = E'\sigma\vartheta = E'\lambda = F'$

entonces E es un reducto de E'' tal que $E \leq_s F$. Pero E es además un factor de C, pues reductos de factores son factores. Lo cual concluye la prueba.

⊣

Lema 2.1 *(Lema del levantamiento).*
Si C, D, C', D' son cláusulas tales que $C \leq_s C'$ y $D \leq_s D'$ y E' es un resolvente binario de C' y D', entonces existe un resolvente binario E de C y D tal que $E \leq_s E'$.

dem:

Supongamos s.p.g. que $V(C) \cap V(D) = \varnothing = V(C') \cap V(D')$.

Por hipótesis existen λ, μ tales que $C\lambda = C'$ y $D\mu = D'$. Sean $C' = C'_1 \vee L' \vee C'_2$ y $D' = D'_1 \vee M' \vee D'_2$, entonces $E' = (C'_1 \vee C'_2 \vee D'_1 \vee D'_2)\sigma$ donde σ es umg de $\{L'^c, M'\}$

Por la forma en que estan dadas C', D' tenemos que $C = C_1 \vee L \vee C_2$ y $D = D_1 \vee M \vee D_2$, donde $L\lambda = L'$, $M\mu = M'$ y $C_i\lambda = C'_i, D_i\mu = D'_i$.
Como $V(C) \cap V(D) = \varnothing$ entonces la sustitución $\eta = \lambda \cup \mu$ está bien definida y cumple $C\eta = C', D\eta = D'$. Además como σ es umg de $\{L'^c, M'\}$ y $L' = L\eta, M' = M\eta$ entonces σ es umg de $\{L^c, M\}\eta$ por lo que $\eta\sigma$ unifica a $\{L^c, M\}$, así que existe un umg de $\{L^c, M\}$ digamos τ.

Como τ es umg entonces existe una sustitución ρ tal que $\tau\rho = \eta\sigma$. Si tomamos $E = (C_1 \vee C_2 \vee D_1 \vee D_2)\tau$ entonces E es un resolvente binario de C y D. Más aún: $E\rho = (C_1 \vee C_2 \vee D_1 \vee D_2)\tau\rho = (C_1 \vee C_2 \vee D_1 \vee D_2)\eta\sigma = (C'_1 \vee C'_2 \vee D'_1 \vee D'_2)\sigma = E'$
$\therefore\ E \leq_s E'$ ⊣

Ahora podemos demostrar el teorema del levantamiento, pieza fundamental para la completud de la resolución.

Teorema 2.2 *(Teorema del levantamiento).*
Sea $\mathbb{S}$ un conjunto de cláusulas y $\mathbb{S}'$ un conjunto de instancias de cláusulas de $\mathbb{S}$. Si Δ es una R-derivación a partir de $\mathbb{S}'$ entonces existe una R-derivación Γ a partir de $\mathbb{S}$ tal que $\Gamma \leq_s \Delta$

dem:

La prueba es por inducción sobra la longitud de la R-derivación Δ denotada $l(\Delta)$.

- $l(\Delta) = 1$.
  En este caso $\Delta = D'$ donde D' es una variante de D y $D \in \mathbb{S}'$, por lo que existen $C \in \mathbb{S}$ y λ tales que $C\lambda = D$. Como D' es variante de D existe una permutación σ tal que $D'\sigma = D$. Así $C\lambda = D'\sigma$ y como σ es permutación entonces σ^{-1} esta bien definida y se tiene que $C\lambda\sigma^{-1} = D'$, es decir $C \leq_s D'$. Por lo que definimos a Γ como C.

- H.I. El teorema es válido si $l(\Delta) \leq n$.
- $l(\Delta) = n + 1$.
  Descomponemos a Δ como $\Delta = \Delta', C.$ donde $l(\Delta') = n$. Tenemos dos casos:

  1. C es variante de D y $D \in \mathbb{S}'$.
     Este caso es inmediato a partir del caso base y de la hipótesis de inducción.
  2. C es resolvente binario de D', E' donde D', E' son factores de cláusulas D, E que figuran en Δ'.

     En este caso podemos expresar a Δ' como $\Delta' = \Delta_1, D, \Delta_2, E, \Delta_3$.

     Mediante la hipótesis de inducción existe una R-derivación Γ' tal que $\Gamma' \leq_s \Delta'$. Entonces Γ' debe ser de la forma $\Gamma' = \Gamma_1, F, \Gamma_2, G, \Gamma_3$ donde $F \leq_s D$, $G \leq_s E$. Ahora bien, dado que D', E' son factores de D, E y D, E son instancias de F, G, como factores de instancias son instancias de factores (teorema 2.1) entonces existen factores F' de F y G' de G tales que $F' \leq_s D'$ y $G' \leq_s E'$.

     Finalmente como C es un resolvente binario de D' y E', mediante el lema del levantamiento obtenemos un resolvente binario H de F', G' tal que $H \leq_s C$. Por lo tanto H es resolvente de F, G. Así que si hacemos $\Gamma = \Gamma', H$ obtenemos que $\Gamma = \Gamma', H \leq_s \Delta', C = \Delta$

⊣

Hasta aquí hemos seguido los métodos presentados en [Lei97]. Sin embargo, para obtener el teorema de completud general, tenemos que partir del teorema de completud para RC-derivaciones, que es el caso particular para cláusulas cerradas, este es una adaptación propia del método presentado en [SA91]. El corolario y la manera de concluir la completud son aportaciónes nuestras.

Definición 2.11 Sean $\mathbb{S}$ un conjunto de cláusulas y C una cláusula. Una RC-derivación de C a partir de $\mathbb{S}$ es una sucesión de cláusulas $C_1, \ldots, C_n$ tal que:

1. $C_n = C$
2. Para toda $i = 1, \ldots, n$, se cumple alguna de las siguientes condiciones:
   (a) C_i es una instancia cerrada de alguna cláusula de $\mathbb{S}$.
   (b) C_i es resolvente de D, E, donde D, E son reductos de C_j, C_k para algunas $j, k < i$

Notación: $\mathbb{S} \underset{RC}{\vdash} C$ significa que existe una RC-derivación de C a partir de $\mathbb{S}$. Si $\mathbb{S} \underset{RC}{\vdash} \square$ decimos que se obtiene una RC-refutación de $\mathbb{S}$.

Obsérvese que la noción de RC-derivación es sobre conjuntos arbitrarios de cláusulas, sin embargo en las derivaciones sólo figuran instancias cerradas, de ahí el nombre de "RC", por Resolución Cerrada.

Para obtener el teorema de completud para las RC-derivaciones utilizaremos una noción equivalente que sólo actúa sobre conjuntos de cláusulas cerradas, llamada Resolución Proposicional (RP).

Definición 2.12 Sean M un conjunto de cláusulas cerradas y C una cláusula. Una RP-derivación de C a partir de M es una sucesión de cláusulas $C_1, \ldots, C_n$ tal que:

1. $C_n = C$
2. Para toda $i = 1, \ldots, n$, se cumple alguna de las siguientes condiciones:
   (a) $C_i \in M$
   (b) C_i se obtuvo de reductos de C_j, C_k donde $j, k < i$ mediante resolución proposicional.[3]

Notación: $M \underset{RP}{\vdash} C$ significa que existe una RP-derivación de C a partir de M. Si $M \underset{RP}{\vdash} \square$ decimos que se obtiene una RP-refutación de M.

Ahora presentamos el teorema de completud para las RP-derivaciones.

Teorema 2.3 *(Completud para RP-derivaciones).*
Sea M un conjunto de cláusulas cerradas. Si M es no satisfacible entonces $M \underset{RP}{\vdash} \square$.

dem:

Podemos suponer que $\square \notin M$ pues de lo contrario el resultado sería trivial. Además utilizando el teorema de Compacidad podemos suponer que M es finito, digamos que $M = \{C_1, \ldots, C_n\}$.

La prueba es mediante inducción sobre el número $k(C_1, \ldots, C_n)$ definido mediante

$$k(C_1, \ldots, C_n) = (\sum_{i=1}^{n} |C_i|) - n.$$

- $k(C_1, \ldots, C_n) = 0$. Lo cual significa que cada cláusula es una literal, $C_i = L_i$, $i = 1, \ldots, n$. Como M no tiene un modelo entonces necesariamente hay un par complementario en M, digamos que $L_j = L_i^c$, $1 \leq i, j \leq n$, el cual produce de inmediato mediante RP la cláusula vacía $\square$.

[3]La resolución proposicional es aquella que no admite unificación, sino que el par complementario de literales tiene que estar presente de entrada.

- H.I. Supongamos válido el resultado para $0 \leq m < k(C_1, \ldots, C_n)$

- $k(C_1, \ldots, C_n) > 0$. En este caso podemos suponer s.p.g. que $|C_1| \geq 2$. Sea L una literal de C_1 y $E = C_1 \setminus L$. Obsérvese que $k(L, C_2, \ldots, C_n) < k(C_1, \ldots, C_n)$, $k(E, C_2, \ldots, C_n) = k(C_1, \ldots, C_n) - 1 < k(C_1, \ldots, C_n)$ y claramente los conjuntos $M_1 = \{L, C_2, \ldots, C_n\}$ y $M_2 = \{E, C_2, \ldots, C_n\}$ no tienen un modelo puesto que $L \vee E = C_1$ y M no tiene un modelo.

  Así que utilizando la hipótesis de inducción obtenemos que $M_1 \underset{RP}{\vdash} \square$ y $M_2 \underset{RP}{\vdash} \square$
  Sea $\Gamma : D_1, \ldots, D_m$ una RP-refutación de M_2 y sea Γ_L la RP-derivación obtenida a partir de Γ sustituyendo D_i por $D_i \vee L$ para $i = 1, \ldots, m$, claramente Γ_L es una derivación de L a partir de M, de manera que $M \underset{RP}{\vdash} L$.

  Por último a partir de $M \underset{RP}{\vdash} \square$ y $M_1 \underset{RP}{\vdash} L$ concluimos que $M \underset{RP}{\vdash} \square$

⊣

Corolario 2.1 *(Completud para RC-derivaciones)*
Si $\mathbb{S}$ *es un conjunto de cláusulas no satisfacible entonces hay una RC-refutación de* $\mathbb{S}$.

dem:

Como $\mathbb{S}$ no tiene un modelo entonces por el teorema de Herbrand 1.3, $IC(\mathbb{S})$ no tiene un modelo. Pero como $IC(\mathbb{S})$ es un conjunto de cláusulas cerradas, por el teorema anterior $IC(\mathbb{S}) \underset{RP}{\vdash} \square$. Pero esto último claramente implica que $\mathbb{S} \underset{RC}{\vdash} \square$ ⊣

Ya podemos presentar el teorema de completud general para R-derivaciones.

Teorema 2.4 *(Completud para R-derivaciones)*
Si $\mathbb{S}$ *es un conjunto no satisfacible de cláusulas entonces hay una R-refutación de* $\mathbb{S}$.

dem:

El corolario anterior nos proporciona una RC-refutación Γ de $\mathbb{S}$. Sea $\mathbb{S}'$ el conjunto de instancias cerradas de $\mathbb{S}$ que figuran en Γ. Como las nociones de *reducto y factor coinciden en cláusulas cerradas entonces*, por definición, Γ es una R-refutación de $\mathbb{S}'$. De manera que, usando el teorema del levantamiento, existe una R-derivación Δ a partir de $\mathbb{S}$ tal que $\Delta \leq_s \Gamma$, pero como $\Delta = \Gamma\sigma$ para alguna substitución σ entonces claramente Δ es una R-refutación de $\mathbb{S}$. ⊣

2.3 Refinamientos de Resolución

En esta sección nos dedicamos a estudiar los refinamientos de resolución, que constituyen una manera de optimizar las derivaciones obtenidas con resolución.

Definición 2.13 Sea $\mathbb{S}$ un conjunto finito de cláusulas. Definimos el conjunto $\Omega(\mathbb{S})$ como el conjunto de todas las R-derivaciones a partir de $\mathbb{S}$, es decir, $\Omega(\mathbb{S}) = \{X \mid X \text{ es una R-derivación a partir de } \mathbb{S}\}$.
Con $\mathfrak{C}$ denotamos al conjunto de todas las cláusulas y con CL denotamos al conjunto de todos los conjuntos finitos de cláusulas. Por último el conjunto de todas las R-derivaciones será denotado con Ω. Claramente se tiene

$$\Omega = \bigcup_{\mathbb{S} \in CL} \Omega(\mathbb{S})$$

Definición 2.14 Considérese una función $\Psi : CL \longrightarrow P(\Omega)$. Ψ es un *refinamiento* de resolución syss se cumplen las siguientes condiciones:

1. Para todo $\mathbb{S} \in CL$, $\Psi(\mathbb{S}) \subseteq \Omega(\mathbb{S})$.
2. El conjunto $\{\Pi \mid \Pi \in \Psi(\mathbb{S})\}$ es decidible para cualquier $\mathbb{S} \in CL$.
3. Existe un algoritmo α que construye $\Psi(\mathbb{S})$ para cada $\mathbb{S} \in CL$. (Nótese que $\Psi(\mathbb{S})$ puede ser infinito).
4. Si $\mathbb{S} \subseteq \mathbb{Q}$ entonces $\Psi(\mathbb{S}) \subseteq \Psi(\mathbb{Q})$

Definición 2.15 Un refinamiento de resolución Ψ se llama *completo* syss para cada conjunto finito no satisfacible de cláusulas $\mathbb{S}$, $\Psi(\mathbb{S})$ contiene una R-refutación de $\mathbb{S}$.

El siguiente teorema proporciona una condición necesaria para que un refinamiento sea completo.

Teorema 2.5 *Sea Ψ un refinamiento. Si Ψ es completo entonces debe existir un conjunto finito de cláusulas $\mathbb{S}$ tal que $\Psi(\mathbb{S})$ es infinito.*

dem:

Supongamos que Ψ es completo y que $\Psi(\mathbb{S})$ es finito para todo $\mathbb{S} \in CL$. Por definición existe un algoritmo α tal que $\alpha(\mathbb{S}) = \Psi(\mathbb{S})$ y como $\Psi(\mathbb{S})$ es finito entonces α siempre termina. Además, como Ψ es completo entonces $\Psi(\mathbb{S})$ contiene una R-refutación syss $\mathbb{S}$ es no satisfacible. De manera que construyendo $\Psi(\mathbb{S})$ mediante α y verificando si $\Psi(\mathbb{S})$ contiene una refutación, obtenemos un procedimiento de decisión para el problema de la satisfacibilidad de la lógica clausular, lo cual es absurdo puesto que si la lógica clausular fuera decidible entonces la lógica de predicados también sería decidible, lo cual es absurdo por

el teorema de indecidibilidad de Church [SA91]. ⊣

Este teorema muestra que el problema de la parada para refinamientos completos siempre es indecidible.
En general una aplicación de un refinamiento completo Ψ lleva a uno de los siguientes resultados:

1. La derivación de $\square$, es decir, $\square \in \Psi(\mathbb{S})$.
2. $\Psi(\mathbb{S})$ es finito y no contiene una refutación de $\mathbb{S}$. Lo cual significa que Ψ , considerado como un proceso, termina en $\mathbb{S}$ y por la completud de Ψ sabemos que $\mathbb{S}$ es satisfacible.
3. $\Psi(\mathbb{S})$ es infinito y no contiene una refutación de $\mathbb{S}$. En este caso Ψ no termina en $\mathbb{S}$ y, por completud, $\mathbb{S}$ es satisfacible. Este caso corresponde, en programación, a entrar en un ciclo sin fin.

Muchos refinamientos, tales como resolución ordenada, resolución semántica y resolución asegurada[4], se pueden especificar mediante operadores conjuntistas. En estos casos el conjunto de todas las derivaciones a partir de un $\mathbb{S} \in CL$ dado, especificado por un refinamiento Ψ, se puede describir mediante el conjunto de todas las cláusulas derivables a partir de $\mathbb{S}$.

Definición 2.16 Sean C, D dos cláusulas. El conjunto de todos los resolventes de C, D será denotado por $Res(C, D)$.

Obsérvese que aunque $Res(C, D)$ puede ser infinito, es fácil ver que el conjunto $Res(C, D)/\sim_v$ de clases de equivalencia bajo la relación "ser variante de" es un conjunto finito. Al usar operadores de resolución es conveniente evitar diferentes variantes de cláusulas en el conjunto de resolventes derivados. Como es inconveniente trabajar con clases de equivalencia, seleccionamos una cláusula de cada clase. Mediante $\varphi(\mathbb{S})$ denotamos las cláusulas representativas de cada clase de $\mathbb{S}/\sim_v$. Podemos usar un método estandar para renombrar variables en las cláusulas que consiste en reemplazar la primera variable por x_1, la segunda por x_2, y así sucesivamente. El operador φ se conoce como el operador de renombre estandar.

A continuación definimos el primer operador conjuntista de resolución:

Definición 2.17 Sean $\mathbb{S}$ un conjunto finito de cláusulas y $n \in \mathbb{N}$. Definimos:

$$\begin{aligned} Res(\mathbb{S}) &= \bigcup\{Res(C, D) \mid C, D \in \mathbb{S}\} \\ S^0_{\varnothing}(\mathbb{S}) &= \varphi(\mathbb{S}) \\ S^{n+1}_{\varnothing}(\mathbb{S}) &= S^n_{\varnothing}(\mathbb{S}) \cup \varphi(Res(S^n_{\varnothing}(\mathbb{S}))) \\ R_{\varnothing}(\mathbb{S}) &= \bigcup_{n\in\omega} S^n_{\varnothing}(\mathbb{S}). \end{aligned}$$

[4]lock resolution.

$R_\varnothing$ es el operador de resolución ilimitada.

$R_\varnothing(\mathbb{S})$ es el conjunto de todas las cláusulas (con variables en forma estandar) derivables mediante R-deducciones a partir de $\mathbb{S}$. Utilizando el operador φ garantizamos, para cada n, la finitud del conjunto $S^n_\varnothing(\mathbb{S})$, que es el conjunto de las cláusulas deducibles mediante una R-derivación de longitud $\leq n$. La finitud del conjunto $R_\varnothing(\mathbb{S})$ (así como la de $\Psi(\mathbb{S})$ para algun refinamiento Ψ) implica la "terminación de la resolución" en $\mathbb{S}$. Tal terminología está justificada puesto que $R_\varnothing(\mathbb{S})$ se puede considerar, de acuerdo a la definición anterior, como un algoritmo definido en CL. Si $R_\varnothing(\mathbb{S})$ es finito entonces el cálculo termina, en otro caso no lo hace.

Definición 2.18 Un *operador de refinamiento* de resolución R_x es una función $R_x : CL \longrightarrow P(\mathfrak{C})$ que satisface las siguientes condiciones: Existe una función $\rho_x : CL \longrightarrow CL$ tal que:

1. $\rho_x(\mathbb{S})$ es un subconjunto finito de $R_\varnothing(\mathbb{S})$, para toda $\mathbb{S} \in CL$.

2. Existe un algoritmo α que calcula $\rho_x(\mathbb{S})$, para toda $\mathbb{S} \in CL$.

R_x se define, via ρ_x, como sigue:

$$\begin{aligned} S^0_x(\mathbb{S}) &= \varphi(\mathbb{S}^*) \\ S^{n+1}_x(\mathbb{S}) &= S^n_x(\mathbb{S}) \cup \rho_x(S^n_x(\mathbb{S})) \\ R_x(\mathbb{S}) &= \bigcup_{n\in\omega} S^n_x(\mathbb{S}) \end{aligned}$$

donde $\mathbb{S}^*$ es un conjunto de cláusulas tal que $\mathbb{S} \sim_{sat} \mathbb{S}^*$. Esta manera de definir el primer estrato $S^0_x(\mathbb{S})$ es una mejora nuestra a la original de [Lei97] pues proporciona mayor claridad y flexibilidad en los operadores que veremos posteriormente.

Definición 2.19 Un operador de refinamiento de resolución R_x es *completo* syss para todo conjunto no satisfacible de cláusulas $\mathbb{S}$, se tiene que $\square \in R_x(\mathbb{S})$.

Es directo probar un resultado análogo al teorema 2.5 para un operador de refinamiento de resolución completo. En este caso debe existir un $\mathbb{S} \in CL$ tal que $R_x(\mathbb{S})$ es infinito.

Definición 2.20 Sea $\mathfrak{R}$ un conjunto de R-derivaciones. Definimos al conjunto $Der(\mathfrak{R})$ como el conjunto de las cláusulas , con variables en forma estandar, derivadas mediante R-deducciones de $\mathfrak{R}$.

Claramente, para cualquier $\mathbb{S} \in CL$ se tiene que $Der(\Omega(\mathbb{S})) = R_{\varnothing}(\mathbb{S})$.
En algunos casos los conjuntos de R-derivaciones no pueden ser descritos simplemente mediante operadores de refinamiento sobre conjuntos de cláusulas, como es el caso de los refinamientos lineales. Pero por otra parte, los operadores de refinamiento siempre definen refinamientos en el sentido de la definición 2.14, como lo asegura el siguiente teorema:

Teorema 2.6 *Si R_x es un operador de refinamiento de resolución entonces existe un refinamiento Ψ tal que, para toda $\mathbb{S} \in CL$:*

$$Der(\Psi(\mathbb{S})) = R_x(\mathbb{S}).$$

dem:

Sea $\mathbb{S} \in CL$ entonces cada $C \in \mathbb{S}$ es también una R-derivación de longitud 1 y $\mathbb{S}$ se puede considerar también como un conjunto de R-derivaciones.
Vamos a definir Ψ mediante saturación de niveles (como los operadores de refinamiento).

- Sea $\Psi^0(\mathbb{S}) = \varphi(\mathbb{S}^*)$.
  Por definición, la función ρ_x es calculable, además $\rho_x(\mathbb{S}^*)$ es un conjunto finito, digamos que $\rho_x(\mathbb{S}^*) = \{C_1, \ldots, C_n\}$ y $\rho_x(\mathbb{S}^*) \subseteq R_{\varnothing}(\mathbb{S}^*)$. Así que existe un algoritmo α (independiente de $\mathbb{S}$) que produce un conjunto de R-derivaciones a partir de $\mathbb{S}$, $\Pi(\mathbb{S}) = \{\Delta_1, \ldots, \Delta_n\}$ tal que $\mathbb{S} \underset{R}{\vdash} C_i$ mediante la derivación Δ_i. Claramente se cumple que $Der(\Pi(\mathbb{S})) = \rho_x(\mathbb{S})$ y Π es calculable.

- Supongamos que $\Psi^n(\mathbb{S})$ esta definido y sea $\Psi^n(\mathbb{S}) = \{\psi_1, \ldots, \psi_{k(n)}\}$. A partir de todas las derivaciones de $\Psi^n(\mathbb{S})$ obtenemos la derivación

  $$\chi(\Psi^n(\mathbb{S})) = \psi_1 \star \ldots \star \psi_{k(n)}$$

  donde $\star$ denota al operador de concatenación.

- Ahora definimos, $\Psi^{n+1}(\mathbb{S}) = \Psi^n(\mathbb{S}) \cup \{\chi(\Psi^n(\mathbb{S})) \star \psi \mid \psi \in \Pi(Der(\Psi^n(\mathbb{S})))\}$

- Por último definimos $\Psi(\mathbb{S}) = \bigcup_{n \in \omega} \Psi^n(\mathbb{S})$.

Observemos que por construcción $\Psi(\mathbb{S})$ es un conjunto de R-derivaciones a partir de $\mathbb{S}$.

Para mostrar que $Der(\Psi(\mathbb{S})) = R_x(\mathbb{S})$, basta mostrar que para toda $\mathbb{S} \in CL$ y para toda $n \in \omega$, se cumple que $Der(\Psi^n(\mathbb{S})) = S_x^n(\mathbb{S})$ y utilizar la definición de $R_x(\mathbb{S})$. Esto lo haremos mediante inducción sobre n:

- Base de la inducción, $n = 0$.

$$Der(\Psi^0(\mathbb{S})) = Der(\varphi(\mathbb{S}^\star)) = \varphi(\mathbb{S}^\star) = S_x^0(\mathbb{S})$$

- Hipótesis de inducción, $Der(\Psi^n(\mathbb{S})) = S_x^n(\mathbb{S})$
- Paso inductivo. Probaremos la doble contención:

  $\supseteq$) Para esto basta ver que los dos uniendos que conforman a $S_x^{n+1}(\mathbb{S})$ estan contenidos en $Der(\Psi^{n+1}(\mathbb{S}))$.

  * Utilizando la HI tenemos:

$$S_x^n(\mathbb{S}) \underset{HI}{=} Der(\Psi^n(\mathbb{S})) \subseteq Der(\Psi^{n+1}(\mathbb{S}))$$

  * Por definición de Π se tiene que $Der(\Pi(S_x^n(\mathbb{S}))) = \rho_x(S_x^n(\mathbb{S}))$. Pero para cada $\psi \in \Pi(S_x^n(\mathbb{S}))$, $\chi(\Psi^n(\mathbb{S})) \star \psi$ es una R-derivación a partir de $\mathbb{S}$ y

$$Der(\chi(\Psi^n(\mathbb{S})) \star \psi) = Der(\psi)$$

  por lo tanto concluimos que $\rho_x(S_x^n(\mathbb{S})) \subseteq Der(\Psi^{n+1}(\mathbb{S}))$.

  $\subseteq$) Por definición se tiene que $S_x^n(\mathbb{S}) \subseteq S_x^{n+1}(\mathbb{S})$, así que usando la HI obtenemos

$$Der(\Psi^n(\mathbb{S})) \subseteq S_x^{n+1}(\mathbb{S}).$$

  Si $\psi \in \Pi(Der(\Psi^n(\mathbb{S})))$ entonces

$$Der(\chi(\Psi^n(\mathbb{S})) \star \psi) = Der(\psi)$$

  Utilizando la HI obtenemos:

$$Der(\psi) \in Der(\Pi(S_x^n(\mathbb{S})))$$

  Ahora bien, por la relación entre Π y ρ_x, obtenemos $Der(\psi) \in S_x^{n+1}(\mathbb{S})$, y finalmente

$$\{\chi(\Psi^n(\mathbb{S})) \star \psi \mid \psi \in \Pi(Der(\Psi^n(\mathbb{S})))\} \subseteq S_x^{n+1}(\mathbb{S}).$$

⊣

2.4 Hiperresolución

2.4.1 Normalización de Cláusulas

Definición 2.21 Una cláusula es *positiva* si todas sus literales son átomos no negados. Es *negativa* si todas sus literales son átomos negados y es *mixta* en cualquier otro caso.

Definición 2.22 Sea $\prec \subseteq \mathfrak{L} \times \mathfrak{L}$ un orden para las literales (positivas antes de negativas y orden lexicográfico en cada grupo). Si C es una cláusula, denotamos con C_o a la cláusula C con sus literales ordenadas mediante $\prec$. Definimos el *operador de ordenación* $N_o : \mathfrak{C} \longrightarrow \mathfrak{C}$ como $N_o(C) = C_o$

Ejemplo 2.1 Si $C = \neg Q(x,y) \vee R(u) \vee R(b) \vee P(a,c)$ entonces $N_o(C) = P(a,c) \vee R(b) \vee R(u) \vee \neg Q(x,y)$.

Definición 2.23 Sean C una cláusula y η una substitución tal que $C\eta$ es una variante de C con variables $v_1, \ldots, v_n$ (ordenadas de izquierda a derecha) y que cumple $V(C\eta) \cap \{x_i \mid i \in \mathbb{N}\} = \varnothing$. Definimos el *operador de normalización de variables* $N_v : \mathfrak{C} \longrightarrow \mathfrak{C}$ como $N_v(C) = C\eta\{v_1/x_1, \ldots, v_m/x_m\}$.

Ejemplo 2.2 Si $C = P(w) \vee P(a) \vee Q(z,y)$ entonces $N_v(C) = N_v(P(w) \vee P(a) \vee Q(z,y)) = P(x_1) \vee P(a) \vee Q(x_2, x_3)$.

Definición 2.24 Sea C una cláusula y C_r el menor reducto de C, es decir, C_r es un reducto de C que ya no es reducible. Definimos el *operador de reducción* $N_r : \mathfrak{C} \longrightarrow \mathfrak{C}$ como $N_r(C) = C_r$

Ejemplo 2.3 Si $C = P(a) \vee Q(b) \vee R(z) \vee Q(b) \vee R(z)$ entonces $N_r(C) = P(a) \vee Q(b) \vee R(z)$.

Definición 2.25 El operador $N_s : \mathfrak{C} \longrightarrow \mathfrak{C}$ definido mediante $N_s = N_r \circ N_v \circ N_o$ se llama el *operador de normalización estandar.*

Ejemplo 2.4 Si $C = \neg Q(x,z) \vee \neg P(a) \vee R(y) \vee \neg P(a) \vee Q(a,z)$ entonces

$$\begin{aligned} N_s(C) &= N_r \circ N_v(Q(a,z) \vee R(y) \vee \neg P(a) \vee \neg P(a) \vee \neg Q(x,z)) \\ &= N_r(Q(a,x_1) \vee R(x_2) \vee \neg P(a) \vee \neg P(a) \vee \neg Q(x_3,x_1)) \\ &= Q(a,x_1) \vee R(x_2) \vee \neg P(a) \vee \neg Q(x_3,x_1) \end{aligned}$$

Definición 2.26 Una cláusula C esta *condensada* si no existe una literal L en C tal que $C \setminus L$ es un factor de C. Definimos

$$\gamma(C) = \begin{cases} C & \text{Si } C \text{ esta condensada.} \\ \gamma(C \setminus L) & \text{Si } L \text{ es la primera literal de } C \text{ tal que } C \setminus L \text{ es factor de } C. \end{cases}$$

el operador $N_c : \mathfrak{C} \longrightarrow \mathfrak{C}$, definido como $N_c(C) = N_s \circ \gamma(C)$, es la *forma normal condensada* o N_c-normalización de C.

Es fácil ver que la N_c-normalización, considerada como regla de inferencia es correcta, es decir, si C es verdadera entonces $N_c(C)$ es verdadera. Es más, para cualquier cláusula C, se tiene que $N_c(C)$ es lógicamente equivalente a C, como lo asegura la siguiente proposición,

Proposición 2.1 *Para toda cláusula C, se tiene que $\forall N_c(C) \equiv \forall C$.*

dem:

Puesto que es obvio que $\forall N_s(C) \equiv \forall C$, basta demostrar que $\forall\gamma(C) \equiv \forall C$ y para esto es suficiente con mostrar que si $C \setminus L$ es un factor de C entonces $\forall(C \setminus L) \equiv \forall C$. Es obvio que $\models \forall(C \setminus L) \rightarrow \forall C$ y además para cualquier factor D de C, se tiene que $\models \forall C \rightarrow \forall D$, de manera que, como $C \setminus L$ es un factor de C, podemos concluir que $\forall(C \setminus L) \equiv \forall C$. ⊣

Ejemplo 2.5 Sea $C = P(x) \vee R(b) \vee P(a) \vee R(z)$, como $R(b) \vee P(a) \vee R(z)$ es un factor de C entonces $\gamma(C) = \gamma(R(b) \vee P(a) \vee R(z))$. Obsérvese que ni $P(a) \vee R(z)$ ni $R(b) \vee R(z)$ son factores de $\gamma(C)$ pero $R(b) \vee P(a)$ es un factor de $R(b) \vee P(a) \vee R(z)$. En consecuencia tenemos $\gamma(R(b) \vee P(a) \vee R(z)) = \gamma(R(b) \vee P(a))$. Como $R(b) \vee P(a)$ no tiene factores, entonces $\gamma(R(b) \vee P(a)) = R(b) \vee P(a)$. Así que concluimos que $\gamma(C) = R(b) \vee P(a)$ y $N_c(C) = P(a) \vee R(b)$.

2.4.2 El Operador de Hiperresolución

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

Ejemplo 2.6 Mediante Hiperresolución con el unificador $\mu = \{x/Tere, y/Pedro, z/Sergio\}$ a partir del conjunto

$$\begin{array}{l} \neg\, Casado(x,y) \;\vee\; \neg\, Madre(x,z) \;\vee\; Padre(y,z) \\ Casado(Tere, Pedro) \;\vee\; Masviejo(Tere, Pedro) \\ Madre(Tere, Sergio) \end{array}$$

aplicando μ obtenemos:

$$Padre(Pedro, Sergio) \;\vee\; Masviejo(Tere, Pedro)$$

A continuación iniciamos el estudio formal de la hiperresolución, hasta mostrar el teorema de completud correspondiente.

Definición 2.27 Sea C una cláusula de la forma $C_1 \vee C_2$ tal que C_1 es positiva y C_2 es negativa, (alguna o ambas podrían ser $\square$). Entonces C esta en *PN-forma*. C_1 (C_2) se llama la parte positiva (negativa) de C y se denota por C^+ (C^-).

Definición 2.28 Sean C, D dos PN-cláusulas tal que D es positiva y C es de la forma $C' \vee \neg Q$ donde Q es un átomo. Sea D' una variante de un factor de D tal que $V(D') \cap V(C) = \varnothing$ y $D' = D_1 \vee P \vee D_2$ donde P es un átomo. Supongase que $\{P, Q\}$ es unificable mediante el umg σ. Entonces $(D_1 \vee D_2 \vee C')\sigma$ es un *RFP-resolvente* (Restricción a Factores Positivos) de C y D.

Obsérvese que cada RFP-resolvente es también un resolvente ordinario en PN-forma (resolviendo D desde la izquierda). La factorización esta restringida a cláusulas positivas y la regla de resolución solo puede aplicarse a la literal que está más a la derecha en la cláusula no positiva.

Ejemplo 2.7 Sea $\mathbb{S} = \{C_1, C_2, C_3, C_4\}$ donde $C_1 = P(x) \vee P(y), C_2 = P(x) \vee \neg R(x), C_3 = \neg R(x) \vee \neg R(y) \vee \neg P(x), C_4 = R(b)$. Los resolventes $\neg R(x) \vee \neg R(y)$ y $P(x) \vee \neg R(x) \vee \neg R(y)$ son RFP-resolventes de C_1 y C_3.

$\neg R(x) \vee \neg R(u) \vee \neg R(v)$ es un resolvente de C_2 y C_3 pero no es un RFP-resolvente, puesto que ni C_2 ni C_3 son positivas.

$\neg R(x)$ es un resolvente de C_1 y C_3, pero no es un RFP-resolvente, pues se ha aplicado la factorización a una cláusula no positiva.

$\neg P(b)$ es un resolvente de C_3 y C_4 pero no es un RFP-resolvente puesto que la literal eliminada en la cláusula no positiva no es la que está más a la derecha. De hecho no hay RFP-resolventes de C_3 y C_4.

La idea en la hiperresolución es definir una sucesión de RFP-resolventes que produzca una cláusula positiva. El paso para la macroinferencia se basa en una tupla de cláusulas que contienen una cláusula no positiva y diversas cláusulas positivas.

Definición 2.29 Sean C una PN-cláusula no positiva y $D_1, \ldots, D_n$ cláusulas positivas. La tupla $\varrho : (C; D_1, \ldots, D_n)$ se llama una *sucesión conflictiva*. Si $\{C, D_1, \ldots, D_n\} \subseteq \mathbb{S}$ decimos que ϱ es una sucesión conflictiva a partir de $\mathbb{S}$. Las cláusula C es el *núcleo* y las D_i son los *satélites* de ϱ.

Definición 2.30 Sea $\varrho : (C; D_1, \ldots, D_n)$ una sucesión conflictiva. Para $i < n$ tomense las siguientes cláusulas: $R_0 = C$ y R_{i+1} un RFP-resolvente de R_i y D_{i+1} (si tal resolvente existe). Si R_n existe y es positiva entonces la forma normal condensada de R_n, $N_c(R_n)$ es un *hiperresolvente* de ϱ.

Ejemplo 2.8 Sea $\mathbb{S} = \{C_1, C_2, C_3\}$ con $C_1 = Q(x, y) \vee \neg P(f(x)) \vee \neg P(g(y))$, $C_2 = P(x) \vee R(x), C_3 = P(y) \vee P(f(y))$.
Sea $\varrho = (C_1; C_2, C_3)$, claramente ϱ es una sucesión conflictiva. Definimos

$R_0 = C_1$ y $R_1 = R(g(y)) \vee Q(x,y) \vee \neg P(f(x))$. Hay 2 posibilidades para definir R_2, a saber: $R_2^1 = P(x) \vee R(g(y)) \vee Q(x,y)$ y $R_2^2 = P(f(f(x))) \vee R(g(y)) \vee Q(x,y)$. Ambas cláusulas definen respectivamente a los hiperresolventes $P(x_1) \vee Q(x_1,x_2) \vee R(g(x_2))$ y $P(f(f(x_1))) \vee Q(x_1,x_2) \vee R(g(x_2))$.

Definición 2.31 Sea $\mathbb{S}$ un conjunto finito de PN-cláusulas. Definimos la función $\rho_H : CL \longrightarrow CL$ como

$$\rho_H(\mathbb{S}) = \{X \in \mathfrak{C} \mid X \text{ es un hiperresolvente de } \varrho \text{ y } \varrho \text{ es una sucesión conflictiva a partir de } \mathbb{S}\}$$

$\rho_H(\mathbb{S})$ es el conjunto de todos los hiperresolventes definidos mediante sucesiones conflictivas a partir de $\mathbb{S}$. Sea $\mathbb{S}^+$ el conjunto de cláusulas positivas de $\mathbb{S}$, definimos el *operador de hiperresolución* R_H como sigue:

$$\begin{aligned} S_H^0(\mathbb{S}) &= \varphi(N_c(\mathbb{S}^+) \cup (\mathbb{S} - \mathbb{S}^+)) \\ S_H^{n+1}(\mathbb{S}) &= S_H^n(\mathbb{S}) \cup \rho_H(S_H^n(\mathbb{S})) \\ R_H(\mathbb{S}) &= \bigcup_{n \in \mathbf{N}} S_H^n(\mathbb{S}). \end{aligned}$$

Obsérvese que realmente $S_H^0(\mathbb{S}) = N_c(\mathbb{S}^+) \cup \varphi(\mathbb{S} - \mathbb{S}^+)$

2.4.3 El Teorema de Completud

Ya estamos preparados para demostrar la completud del refinamiento de hiperresolución, para lo cual demostraremos primero, la completud para cláusulas cerradas, utilizando en parte el método de Davis-Putnam.

Lema 2.2 *(Completud de la hiperresolución para cláusulas cerradas)*
Sea $\mathbb{S}$ un conjunto no satisfacible de PN-cláusulas cerradas entonces $\square \in R_H(\mathbb{S})$.

dem:

Podemos suponer s.p.g. que $\mathbb{S}$ es finito. La demostración será por inducción sobre el número de presencias de literales en $\mathbb{S}$, denotado $npl(\mathbb{S})$

- Base de la inducción, $npl(\mathbb{S}) = 0$. En este caso $\mathbb{S} = \{\square\}$ y claramente $\square \in S_H^0(\mathbb{S}) \subset R_H(\mathbb{S})$.
- Hipótesis de inducción: para $npl(\mathbb{S}) \leq n$
- Paso inductivo: Supongamos que $npl(\mathbb{S}) = n + 1$. Hay que analizar dos casos:

1. Todas las cláusulas de $\mathbb{S}$ son positivas o negativas y todas las cláusulas positivas son unitarias[5]. Como $\mathbb{S}$ es no satisfacible, debe haber una cláusula negativa $D \in \mathbb{S}$ tal que $D = \neg P_1 \vee \ldots \vee \neg P_n$ donde las cláusulas unitarias $P_1, \ldots, P_n$ pertenecen a $\mathbb{S}$. Si tal cláusula no existiera todas las cláusulas negativas podrían eliminarse mediante la regla de la literal pura de DP; las cláusulas restantes serían positivas y $\mathbb{S}$ sería satisfacible. Por lo tanto tal D existe y la sucesión $(\neg P_1 \vee \ldots \vee \neg P_n; P_n, \ldots, P_1)$ es una sucesión conflictiva cuyo hiperresolvente es $\square$. De manera que, por definición de R_H, concluimos que $\square \in R_H(\mathbb{S})$.
2. Hay cláusulas mixtas en $\mathbb{S}$ o hay una cláusula positiva no unitaria en $\mathbb{S}$.
   (a) En $\mathbb{S}$ figura una cláusula mixta, digamos C.
   Como C es una PN-cláusula debe ser de la forma $P \vee E$, donde spg P es un átomo. Como $\mathbb{S}$ es no satisfacible, es fácil ver que los conjuntos de cláusulas $\mathbb{S}_1 = (\mathbb{S} - \{C\}) \cup \{P\}$ y $\mathbb{S}_2 = (\mathbb{S} - \{C\}) \cup \{E\}$ son ambos no satisfacibles y $npl(\mathbb{S}_1), npl(\mathbb{S}_2) \leq n$ de manera que $\square \in R_H(\mathbb{S}_1)$ y $\square \in R_H(\mathbb{S}_2)$.
   (b) Existe una cláusula positiva no unitaria $D \in \mathbb{S}$. Entonces D es de la forma $P \vee E$, con E una cláusula positiva, analogamente al caso anterior, descomponemos a $\mathbb{S}$ en $\mathbb{S}_1$ y $\mathbb{S}_2$ obteniendo mediante la HI que $\square \in R_H(\mathbb{S}_1)$ y $\square \in R_H(\mathbb{S}_2)$.

Ahora mostraremos que a lo más la diferencia entre $R_H(\mathbb{S})$ y $R_H(\mathbb{S}_2)$ se dá en las presencias del átomo P en algunas cláusulas. Para un tratamiento formal definimos, para cualquier literal L, una relación $\leq_L$ como sigue:

$$C \leq_L D \text{ syss } N_c(C) = N_c(D) \text{ ó } N_c(D) = N_c(C \vee L)$$

Dejamos como ejercicio el cerciorarse que $(\mathfrak{C}, \leq_L)$ es un preorden. Si $\mathcal{P}, \mathcal{Q}$ son conjuntos de cláusulas, decimos que $\mathcal{P} \leq_L \mathcal{Q}$ syss para toda cláusula $C \in \mathcal{P}$ existe una cláusula $D \in \mathcal{Q}$ tales que $C \leq_L D$. Utilizando este orden parcial mostraremos en ambos casos

$$R_H(\mathbb{S}_2) \leq_P R_H(\mathbb{S}).$$

Para demostrar la relación sobre los estratos de R_H, procedemos por inducción sobre los naturales, es decir, vamos a demostrar que

$$S^n_H(\mathbb{S}_2) \leq_P S^n_H(\mathbb{S}) \text{ para toda } n \in \mathbb{N}$$

– Base de la inducción, $n = 0$: las relaciones

$$N_c(\mathbb{S}_2^+) \leq_P N_c(\mathbb{S}^+) \text{ y } (\mathbb{S}_2 - \mathbb{S}_2^+) \leq_P (\mathbb{S} - \mathbb{S}^+)$$

son triviales puesto que $E \leq_P P \vee E$.

[5] recuérdese que una cláusula unitaria es aquella que contiene exactamente una literal.

Es fácil ver que el paso inductivo se reduce a lo siguiente: Sean $\gamma_1 = (E; D_1, \ldots, D_n)$ y $\gamma_2 = (E'; D'_1, \ldots, D'_n)$ sucesiones conflictivas tales que $E \leq_P E'$ y $D_i \leq_P D'_i$ para $i = 1, \ldots, n$ y las D_i, D'_i están en forma normal condensada. Si F es un hiperresolvente de γ_1 entonces existe un hiperresolvente F' de γ_2 tal que $F \leq_P F'$.
La última aseveración puede todavía reducirse a la preservación de la relación $\leq_P$ en los pasos internos al resolver la sucesión conflictiva. Un resolvente interno es un resolvente de las cláusulas $E_1 \vee \neg Q$ y D_i, donde D_i es positiva y de la forma $F_1 \vee Q \vee F_2$. y el preservar la relación significa que: si $E_1 \vee \neg Q \leq_P G$ y $D_i \leq_P H$ entonces si K es un RFP-resolvente de $E_1 \vee \neg Q$ y D_i y K' es un RFP-resolvente de G y H, entonces $K \leq_P K'$.

Tomemos pues $E_1 \vee \neg Q \leq_P G$ y $F_1 \vee Q \vee F_2 = D_i \leq_P H$. Como P es positiva y G esta en PN-forma tenemos que $G = E'_1 \vee \neg Q$ para alguna cláusula E'_1 tal que $E_1 \leq_P E'_1$. La cláusula H debe contener a Q y es de la forma $F'_1 \vee Q \vee F'_2$ donde $F_1 \vee F_2 \leq_P F'_1 \vee F'_2$ (todas las relaciones se dan sin importar si P es o no Q).

El RFP-resolvente de $E_1 \vee \neg Q$ y $F_1 \vee Q \vee F_2$ es $F_1 \vee F_2 \vee E_1$, ya que las cláusulas positivas están en N_c-forma, por lo que no necesitamos reducción. Análogamente $F'_1 \vee F'_2 \vee E'_1$ es RFP-resolvente de $E'_1 \vee \neg Q$ y $F'_1 \vee Q \vee F'_2$. Porque, en general, $C_1 \leq_P C_2$ y $D_1 \leq_P D_2$ implica que $C_1 \vee D_1 \leq_P C_2 \vee D_2$, podemos concluir que $F_1 \vee F_2 \vee E_1 \leq_P F'_1 \vee F'_2 \vee E'_1$.

De manera que podemos concluir que, para los hiperresolventes F, F' de γ_1 y γ_2 que $F \leq_P F'$.

Estando completa la inducción, utilizando la definición de R_H podemos concluir que

$$R_H(\mathbb{S}_2) \leq_P R_H(\mathbb{S})$$

Es importante observar que el agregar literales positivas a una cláusula no bloquea las posibilidades de pasos de resolución antes existentes.

Por último tenemos que concluir que $\square \in R_H(\mathbb{S})$. Por HI sabemos que $\square \in R_H(\mathbb{S}_2) \leq_P R_H(\mathbb{S})$, de manera que, por definición de $\leq_P$, debe existir una cláusula $C \in R_H(\mathbb{S})$ tal que $\square \leq_P C$, lo cual implica dos casos:

1. $C = \square$, por lo que inmediatamente $\square \in R_H(\mathbb{S})$.
2. $C = P$, puesto que las cláusulas positivas de $R_H(\mathbb{S})$ estan en N_c-forma.
   Ahora obsérvese lo siguiente:

   $$\mathbb{S}_1 = (\mathbb{S} - \{C\}) \cup \{P\} \text{ y } \mathbb{S} - \{C\} \subseteq \mathbb{S} \subseteq R_H(\mathbb{S});$$

   Como $P \in R_H(\mathbb{S})$, obtenemos $\mathbb{S}_1 \subseteq R_H(\mathbb{S})$, de manera que, por la monotonía e idempotencia de R_H concluimos que $R_H(\mathbb{S}_1) \subseteq R_H(\mathbb{S})$. Así que $\square \in R_H(\mathbb{S})$, puesto que, por HI, $\square \in R_H(\mathbb{S}_1)$

Con esto termina la inducción sobre $npl(\mathbb{S})$ y la prueba misma. ⊣

Definición 2.32 Sean C,D dos cláusulas. Definimos la relación $\leq_{sc}$ como: $C \leq_{sc} D$ syss existe σ tal que $N_c(C\sigma) = D$.

Observemos que para cualquier cláusula C y cualquier σ, se cumple que $C \leq_{sc} N_c(C\sigma)$.

Definición 2.33 Sean C,D dos PN-cláusulas. Definimos la relación $\leq_{scs}$ como: $C \leq_{scs} D$ syss existe σ tal que $N_c(C^+\sigma) = N_c(D^+)$ y $C^-\sigma = D^-$.

Lema 2.3 *Sean $C' = C'_1 \vee \neg Q'$, $D' = D'_1 \vee Q' \vee D'_2$ dos PN-cláusulas cerradas, D' en N_c-forma y positiva. Sea $E' = D'_1 \vee D'_2 \vee C'_1$ el RFP-resolvente de C' y D'. Si C, D son dos cláusulas con variables ajenas tales que $C \leq_{scs} C'$ y $D \leq_{sc} D'$ entonces existe un RFP-resolvente E, de C y D tal que $E \leq_{scs} E'$*

dem:

Primero veamos que C debe ser de la forma $C_1 \vee \neg Q$, con $C_1 \leq_{scs} C'_1$ y $Q \leq_s Q'$. $C \leq_{scs} C' \Rightarrow N_c(C^+\sigma) = N_c((C'_1 \vee \neg Q')^+)$ y $C^-\sigma = (C'_1 \vee \neg Q')^-$ para alguna σ. Obsérvese que $C'^+ = C'^+_1$ y $C'^- = C'^-_1 \vee \neg Q'$ de manera que $C^-\sigma = C'^-_1 \vee \neg Q'$, de donde se obtiene que deben existir C'' y Q tales que $Q\sigma = Q'$, es decir, $Q \leq_s Q'$ y $C^- = C''^- \vee \neg Q$.
De manera que $C = C^+ \vee C^- = C^+ \vee C''^- \vee \neg Q$, y si tomamos $C_1 = C^+ \vee C''^-$, es facil ver que $C_1 \leq_{scs} C'_1$.
Ahora bien, $D \leq_{sc} D'$ implica que existe η tal que $N_c(D\eta) = D'$ y como $V(C) \cap V(D) = \varnothing$ entonces la substitución $\mu = \sigma \cup \eta$ esta bien definida y cumple las siguientes 3 condiciones: $N_c(C^+\mu) = N_c(C'^+)$, $C^-\mu = C'^-$ y $N_c(D\mu) = D'$.
De esta última condición, como $D' = D'_1 \vee Q' \vee D'_2$ entonces D, debe tener, la siguiente forma, $D = D_1 \vee L_1 \ldots \vee D_n \vee L_n \vee D_{n+1}$, donde $L_1, \ldots, L_n$ son literales, tales que $\mathbb{L}\mu = \{L_1, \ldots, L_n\}\mu = \{Q'\}$ y $N_c((D_1 \vee \ldots \vee D_{n+1})\mu) = D'_1 \vee D'_2$.
Como $\mathbb{L}$ es unificable, tomemos un umg de $\mathbb{L}$, digamos ρ. Tal ρ define al factor $D_1\rho \vee L_1\rho \vee \ldots \vee D_n\rho \vee D_{n+1}\rho$ de D. Es facil ver que μ unifica a $\{L_1\rho, Q\}$, de manera que con el factor dado de D y C obtenemos el RFP-resolvente $E = ((D_1 \vee \ldots \vee D_{n+1})\rho \vee C_1)\theta$, donde θ es un umg de $\{L_1\rho, Q\}$.
Por último veamos que $E \leq_{scs} E'$. Obsérvese que $\rho\theta \leq_s \mu$ pues $\rho\theta$ es umg de $\mathbb{L} \cup \{Q\}$.
Como ρ define una G-instancia de D y $V(C) \cap V(D) = \varnothing$ entonces $dom\rho \cap V(D) = \varnothing$, por lo que $C_1\rho = C_1$ y $E = (D_1 \vee \ldots \vee D_{n+1} \vee C_1)\rho\theta$.
Consideremos ahora la parte negativa de E. $E^- = C_1^-\rho\theta$, puesto que las D_i son positivas. De la misma manera, concluimos que $E'^- = C'^-_1$. Observemos que $C_1^-\mu = C'^-_1 = E'^-$, puesto que $C^-\mu = C'^-$.
Por otra parte, $E^+ = (D_1 \vee \ldots \vee D_{n+1} \vee C_1^+)\rho\theta$. Mediante las relaciones $N_c(C^+\mu) = N_c(C'^+)$ y $N_c((D_1 \vee \ldots \vee D_{n+1})\mu) = D'_1 \vee D'_2$ obtenemos que $N_c((D_1 \vee \ldots \vee D_{n+1} \vee C_1^+)\mu) = N_c(D'_1 \vee D'_2 \vee C'^+_1)$.

Finalmente, sea τ tal que $\rho\theta\tau = \mu$, entonces, a partir de lo arriba demostrado, es fácil ver que, $N_c(E^+\tau) = N_c(E'^+)$ y $E^-\tau = E'^-$, es decir, $E \leq_{scs} E'$. ⊣

Lema 2.4 *Sea $\mathbb{S}$ un conjunto de cláusulas tal que $N_c(\mathbb{S}^+) = \mathbb{S}^+$ y $\mathbb{S}' \subseteq IC(\mathbb{S})$. Para cada cláusula en N_c-forma $C' \in R_H(\mathbb{S}')$ existe una cláusula $C \in R_H(\mathbb{S})$ tal que $C \leq_{sc} C'$. Tal propiedad la denotamos con $R_H(\mathbb{S}) \leq_{sc} R_H(\mathbb{S}')$.*

dem:

La demostración se hará por inducción sobre n, para los estratos de R_H.

- Base de la inducción, $n = 0$. P.D. $S^0_H(\mathbb{S}) \leq_{sc} S^0_H(\mathbb{S}')$.
  Para esto, dado que las cláusulas de $\mathbb{S}^+$ ya están en N_c-forma y que el operador φ no afecta la forma de las cláusulas, basta mostrar que $\mathbb{S} \leq_{sc} N_c(\mathbb{S}'^+) \cup (\mathbb{S}' - \mathbb{S}'^+)$. Pero esto último es consecuencia inmediata de que $C \leq_{sc} N_c(C\sigma)$ para cualquier substitución σ y $C \in \mathbb{S}$.

- HI: $S^n_H(\mathbb{S}) \leq_{sc} S^n_H(\mathbb{S}')$

- Paso inductivo, P.D. $S^{n+1}_H(\mathbb{S}) \leq_{sc} S^{n+1}_H(\mathbb{S}')$
  Por la definición de S^{n+1}_H, utilizando la hipótesis de inducción, solamente basta demostrar que $\rho_H(S^n_H(\mathbb{S})) \leq_{sc} \rho_H(S^n_H(\mathbb{S}'))$.
  Sea $E' \in \rho_H(S^n_H(\mathbb{S}'))$, entonces por definición de ρ_H, existe una sucesión conflictiva, digamos $\gamma_2 = (C'; D'_1, \ldots, D'_n)$, de elementos de $S^n_H(\mathbb{S}')$ tal que E' es un hiperresolvente de γ_2. Utilizando la HI, obtenemos una sucesión conflictiva $\gamma_1 = (C; D_1, \ldots, D_n)$ de elementos de $S^n_H(\mathbb{S})$ tal que $C \leq_s C'$ y $D_i \leq_{sc} D'_i$ para cualquier $i \leq n$.[6] Es fácil ver que $C \leq_s C' \Rightarrow C \leq_{scs} C'$, por lo que podemos aplicar el lema 2.3 a las cláusulas C', D'_1, R'_1, siendo R'_1 el RFP-resolvente de C' y D'_1 ; con lo cual obtenemos un RFP-resolvente R_1, de C y D_1 tal que $R_1 \leq_{sc} R'_1$. Iterando este proceso n veces, obtenemos que $R_n \leq_{sc} R'_n$, pero, por definición, R_n es el hiperresolvente de γ_2, de manera que, haciendo $E = R_n$ obtenemos que E es un hiperresolvente de γ_1, es decir, $E \in \rho_H(S^n_H(\mathbb{S}))$ y además $E \leq_{sc} E'$.

⊣

La aparente restricción $N_c(\mathbb{S}^+) = \mathbb{S}^+$ impuesta en las hipótesis del lema anterior resulta inofensiva pues, por la proposición 2.1, $N_c(\mathbb{S}^+) \equiv \mathbb{S}^+$. Sin embargo el lema se enuncia de esta manera por motivos de claridad en la demostración. Con los lemas anteriores podemos concluir de inmediato la completud para la hiperresolución.

Teorema 2.7 *(Completud de la Hiperresolución).*
Si $\mathbb{S}$ es un conjunto no satisfacible de PN-cláusulas entonces $\square \in R_H(\mathbb{S})$.

[6] $C \leq_s C'$ no por la HI sino porque $S^n_H(\mathbb{S}')$ se generó a partir de instancias cerradas de $\mathbb{S}$.

dem:

Podemos suponer spg que $N_c(\mathbb{S}^+) = \mathbb{S}^+$. Utilizando los teoremas de Herbrand y de Compacidad, tomemos un conjunto finito de instancias cerradas de $\mathbb{S}$, digamos $\mathbb{S}' \subseteq IC(\mathbb{S})$ tal que $\mathbb{S}'$ es no satisfacible. Por el lema 2.2 podemos concluir que $\square \in R_H(\mathbb{S}')$. Por el lema 2.4, tenemos $R_H(\mathbb{S}) \leq_{sc} R_H(\mathbb{S}')$, de manera que existe una cláusula $C \in R_H(\mathbb{S})$ tal que $C \leq_{sc} \square$, pero esto sólo puede suceder si $C = \square$. Por lo tanto $\square \in R_H(\mathbb{S})$. ⊣

2.5 Hiperresolución Negativa

Definición 2.34 Sea $\mathcal{P} = \{P_1, \ldots, P_n\}$ un conjunto de símbolos de predicado y γ una función con dominio $\mathcal{P}$ tal que para toda $i = 1, \ldots, n$, $\gamma(P_i) = P_i$ o $\gamma(P_i) = \neg P_i$. Entonces γ es un *cambio de signos*. Denotamos a γ mediante su imagen, $\gamma = \{\gamma(P_1), \ldots, \gamma(P_n)\}$. Un cambio de signos γ se puede extender a cláusulas y a conjuntos de cláusulas de la siguiente manera:

$$\begin{aligned}
\gamma(P(t_1, \ldots, t_n)) &= \gamma(P)(t_1, \ldots, t_n) && \text{para fórmulas atómicas} \\
\gamma(\neg P(t_1, \ldots, t_n)) &= \gamma(P(t_1, \ldots, t_n))^c && \text{para literales negativas} \\
\gamma(C_1 \vee \ldots \vee C_n) &= \gamma(C_1) \vee \ldots \vee \gamma(C_n) && \text{para cláusulas} \\
\gamma(\mathbb{S}) &= \{\gamma(C) \mid C \in \mathbb{S}\} && \text{para conjuntos de cláusulas}
\end{aligned}$$

Es fácil ver que los cambios de signos preservan la satisfacibilidad, es decir, que para todo conjunto de cláusulas $\mathbb{S}$, $\mathbb{S} \sim_{sat} \gamma(\mathbb{S})$. Si definimos el conjunto de literales $SL(\mathbb{S})$ de un conjunto de cláusulas $\mathbb{S}$, como $SL(\mathbb{S}) = SP(\mathbb{S}) \cup \{\neg P \mid P \in SP(\mathbb{S})$ y $\neg P$ figura en $\mathbb{S}\}$, donde $SP(\mathbb{S})$ denota al conjunto de símbolos de predicado que figuran en $\mathbb{S}$, entonces la hiperresolución, como la hemos definido, es un principio que genera solamente cláusulas C tales que $SL(C) \subseteq SP(C)$.
El principio de hiperresolución puede generalizarse al substituir en las definiciones concernientes, el concepto de cláusula positiva por el concepto de cláusula γ-positiva, que definimos a continuación.

Definición 2.35 Sea $\mathbb{S}$ un conjunto de cláusulas y γ un cambio de signos para $SP(\mathbb{S}) = \{P_1, \ldots, P_n\}$. Decimos que una cláusula C es *γ-positiva* si $SL(C) \subseteq \{\gamma(P_1), \ldots, \gamma(P_n)\}$.

En particular si $\gamma_1 = SP(\mathbb{S})$ entonces una cláusula γ_1-positiva es simplemente una cláusula positiva y si $\gamma_2 = \{\neg P_1, \ldots, \neg P_n\}$, entonces las cláusulas γ_2-positivas son las cláusulas negativas.
Al tener bien determinado el concepto de cláusula γ-positiva obtenemos, para cada cambio de signo γ, el principio de γ-hiperresolución. En particular, para el

γ_1 anterior la γ_1-hiperresolución es simplemente la hiperresolución y para el γ_2 anterior se obtiene el principio de hiperresolución negativa que simplemente consiste en intercambiar los papeles de las cláusulas positivas y negativas en el principio de hiperresolución. La completud de los principios de γ-hiperresolución, para cualquier γ, se sigue inmediatamente de la completud de la hiperresolución positiva, que ya demostramos, modificando las definiciones y el operador R_H de manera adecuada.

2.6 UR-Resolución

En esta sección presentamos el refinamiento de UR-resolución, el cual es más simple que la hiperresolucion y suele presentarse antes que ésta, sin embargo aquí lo tratamos como un caso particular de γ-hiperresolución por lo que lo presentamos hasta ahora.
Durante algún tiempo el problema central en el área de razonamiento automático fue el desarrollo de estrategias de restricción y la investigación de sus propiedades.
Una de las estrategias de restricción más simples es la llamada *resolución unitaria*, la cual prohibe la generación de resolventes a partir de dos cláusulas no unitarias, es decir, se exige que cada resolvente tenga como padre al menos una cláusula unitaria. Esta estrategia siempre produce resolventes con menos literales que el más grande de los padres.

Definición 2.36 Sean C, D dos cláusulas y E un resolvente de C y D. Decimos que E se obtuvo mediante *resolución unitaria o U-Resolución*, o que E es un *U-resolvente* de C y D si al menos uno de los factores de C y D, utilizados para obtener E, es una cláusula unitaria.
Una *U-derivación* es una R-derivación donde cada resolvente es un U-resolvente.
Una *U-refutación* de un conjunto $\mathbb{S}$ es una U-derivación de $\square$ a partir de $\mathbb{S}$.

El principio de U-resolución resulta incompleto. Por ejemplo, el conjunto $\mathbb{S} = \{Pa \vee Qa,\ \neg Pa \vee Qa,\ Pa \vee \neg Qa,\ \neg Pa \vee \neg Qa\}$ es no satisfacible y sin embargo no existe una U-refutación de $\mathbb{S}$.
Aun cuando la U-resolución es incompleta, es un principio eficiente y suficientemente poderoso como para probar una gran clase de teoremas, o lo que es lo mismo, para refutar una gran clase de conjuntos que incluye a todos los conjuntos de cláusulas de Horn. A la clase de conjuntos refutables mediante U-resolución, se le llama *clase de conjuntos U-refutables.*

La resolución que resulta en cláusulas unitarias o UR-resolución es un caso particular de γ-hiperresolución donde los satélites son cláusulas unitarias.

Definición 2.37 Sean C una cláusula y $D_1, \ldots, D_k$ cláusulas unitarias. Considérense las siguientes cláusulas: $R_0 = C$ y R_{i+1} un U-resolvente de R_i y D_{i+1}, si tal resolvente existe e $i < k$.

Si R_k existe y es una cláusula unitaria entonces la sucesión $\varrho : (C; D_1, \ldots, D_k)$ es un *UR-conflicto* y $N_c(R_k)$ es un *UR-resolvente* de ϱ.

Es claro que el principio de UR-resolución es incompleto puesto que es una macroinferencia basada en U-resolución.
De manera análoga a la construcción de R_H podemos construir el operador de UR-resolución R_{UR} y adaptando la prueba de completud para la hiperresolución podemos obtener el siguiente teorema de completud para la UR-resolución restringida a conjuntos U-refutables.

Teorema 2.8 *Sea* $\mathbb{S}$ *un conjunto U-refutable y no satisfacible entonces* $\square \in R_{UR}(\mathbb{S})$.

2.7 Resolución Semántica y el Conjunto de Soporte.

2.7.1 Resolución Semántica

Definición 2.38 Sea $\mathbb{S}$ un conjunto de cláusulas y $\mathfrak{A}$ una interpretación de $\mathbb{S}$. Sean C, D cláusulas de la signatura de $\mathbb{S}$ tales que al menos una de ellas es falsa en $\mathfrak{A}$. Entonces cada resolvente de C y D es un $\mathfrak{A}$-*resolvente*.

El siguiente lema, es de gran importancia para la completud de la resolución semántica y es una consecuencia inmediata del teorema del levantamiento.

Lema 2.5 *Si* D_1, D_2 *son instancias cerradas de* C_1, C_2 *y* F *es un* $\mathfrak{A}$*-resolvente de* D_1, D_2 *entonces existe un* $\mathfrak{A}$*-resolvente* E *de* C_1, C_2 *tal que* $E \leq_s F$.

dem:

En realidad el teorema del levantamiento proporciona un resolvente E de C_1, C_2 tal que $E \leq_s F$, así que lo único que hay que mostrar es que E es un $\mathfrak{A}$-resolvente. Como F es un $\mathfrak{A}$-resolvente, entonces spg D_1 es falsa en $\mathfrak{A}$, y como D_1 es una instancia cerrada de C_1 entonces la fórmula $\forall C_1 \rightarrow D_1$[7] es válida, lo cual implica que C_1 es falsa en $\mathfrak{A}$. Por lo tanto E es un $\mathfrak{A}$-resolvente. $\dashv$

Definición 2.39 Sean $\mathfrak{A}$ una interpretación de un conjunto finito de cláusulas $\mathbb{S}$ y $C, D_1, \ldots, D_k$ cláusulas de la signatura de $\mathbb{S}$, tales que para toda $j = 1, \ldots, k$, D_j es falsa en $\mathfrak{A}$. Para $i < k$ tómense las siguientes cláusulas: $R_0 = C$ y R_{i+1} un $\mathfrak{A}$-resolvente de R_i y D_{i+1} (si tal resolvente existe).
Si R_k existe y es falso en $\mathfrak{A}$ entonces la sucesión $\varrho : (C; D_1, \ldots, D_k)$ se llama un *conflicto semántico* y R_k es un *resolvente semántico* de ϱ con respecto a $\mathfrak{A}$.

[7] $\forall\alpha$ denota a la cerradura universal de la fórmula α.

Ejemplo 2.9 Sean $C = \neg R(f(x)) \vee \neg P(g(x))$, $D_1 = \neg P(x) \vee R(x)$, $D_2 = P(f(x))$. Tomemos $\mathbb{S} = \{C, D_1\}$ y $\mathfrak{A}$ una interpretación tal que $|\mathfrak{A}| = \{0,1\}$, $P^{\mathfrak{A}} = \{1\}$, $R^{\mathfrak{A}} = \varnothing$, $f^{\mathfrak{A}} = \{(0,0),(1,0)\}$, $g^{\mathfrak{A}} = \{(0,1),(1,1)\}$.
En este caso tenemos que $(C; D_1, D_2)$ es un conflicto semántico cuyo resolvente semántico es $\neg P(g(x))$. Note que $(D_1; D_2)$ también es un conflicto semántico porque su resolvente es $R(f(x))$ que es falso en $\mathfrak{A}$, en cambio $(C; D_1)$ no es un conflicto semántico.
Obsérvese que el núcleo de un conflicto puede ser falso en $\mathfrak{A}$, pero debe tener "instancias verdaderas", es decir, debe ser satisfacible.

Definición 2.40 Sean $\mathbb{S}$ un conjunto de cláusulas, $\mathfrak{A}$ una interpretación de $\mathbb{S}$ y $\mathbb{D}$ un conjunto de cláusulas de la signatura de $\mathbb{S}$. Definimos $\rho_{\mathbb{S},\mathfrak{A}}$ como:

$$\rho_{\mathbb{S},\mathfrak{A}}(\mathbb{D}) = \varphi(\{E \mid E \text{ es un resolvente semántico a partir de } \mathbb{D}\})$$

donde φ es el operador de renombre estandar. Definimos de la manera usual el *operador de resolución semántica* como:

$$\begin{aligned} S^0_{\mathbb{S},\mathfrak{A}}(\mathbb{D}) &= \varphi(N_c(\mathbb{D}^+) \cup (\mathbb{D} - \mathbb{D}^+)) \\ S^{n+1}_{\mathbb{S},\mathfrak{A}}(\mathbb{D}) &= S^n_{\mathbb{S},\mathfrak{A}}(\mathbb{D}) \cup \rho_{\mathbb{S},\mathfrak{A}}(S^n_{\mathbb{S},\mathfrak{A}}(\mathbb{D})) \\ R_{\mathbb{S},\mathfrak{A}}(\mathbb{D}) &= \bigcup_{n\in\mathbb{N}} S^n_{\mathbb{S},\mathfrak{A}}(\mathbb{D}). \end{aligned}$$

Definición 2.41 Sean $\mathbb{S} \subset \mathfrak{C}$, $\mathfrak{A}$ una interpretación de $\mathbb{S}$ y $\mathbb{D} \subset IC(\mathbb{S})$. Sea $\mathcal{A}$ el conjunto de átomos de $\mathbb{D}$. Definimos un cambio de signos $\gamma_{\mathbb{D}}$ con dominio $\mathcal{A}$ como $\gamma_{\mathbb{D}}(A) = A$ si y sólo si A es falso en $\mathfrak{A}$. $\gamma_{\mathbb{D}}$ es el *cambio de signos asociado a* $\mathbb{D}$.

Es claro que si $\mathbb{D}$ es no satisfacible, entonces $\gamma_{\mathbb{D}}(\mathbb{D})$ es no satisfacible. Además, se cumple la siguiente propiedad que nos será de gran utilidad: para toda cláusula $D \in \mathbb{D}$, D es falsa en $\mathfrak{A}$ syss $\gamma_{\mathbb{D}}(D)$ es positiva.
La completud de la resolución semántica se reducirá a la completud para la hiperresolución, mediante el siguiente lema

Lema 2.6 *Para cualesquiera* $\mathbb{S}$, $\mathfrak{A}$ *y* $\mathbb{D} \subset IC(\mathbb{S})$ *se cumple que* $\gamma_{\mathbb{D}}(R_{\mathbb{S},\mathfrak{A}}(\mathbb{D})) = R_H(\gamma_{\mathbb{D}}(\mathbb{D}))$.

dem:

Basta probarlo para los estratos correspondientes mediante inducción sobre n.
Sea $\gamma = \gamma_{\mathbb{D}}$.

- Base de la inducción. $n = 0$, es inmediato a partir de $\gamma(N_c(\mathbb{D}^+)) = N_c(\gamma(\mathbb{D})^+)$ y $\gamma(\mathbb{D} - \mathbb{D}^+) = \gamma(\mathbb{D}) - \gamma(\mathbb{D}^+)$.
- Hipótesis de inducción. $\gamma(S^n_{\mathbf{S},\mathfrak{A}}(\mathbb{D})) = S^n_H(\gamma(\mathbb{D}))$
- Paso inductivo. Por las definiciones correspondientes y utilizando la HI, basta demostrar que $\gamma(\rho_{\mathbf{S},\mathfrak{A}}(S^n_{\mathbf{S},\mathfrak{A}}(\mathbb{D}))) = \rho_H(S^n_H(\gamma(\mathbb{D})))$

  $\supseteq$) Sea $R \in \rho_H(S^n_H(\gamma(\mathbb{D})))$, entonces existen cláusulas $C, D_1, \ldots, D_m \in S^n_H(\gamma(\mathbb{D}))$ tales que R es hiperresolvente de $\varrho : (C; D_1, \ldots, D_m)$.
  Por la HI existen cláusulas $E_1, \ldots, E_m \in S^n_{\mathbf{S},\mathfrak{A}}(\mathbb{D})$ tales que $\gamma(E_i) = D_i$ y como $D_i = \gamma(E_i)$ es positiva entonces, dado que E_i se generó a partir de $\mathbb{D}$, concluimos que E_i es falsa en $\mathfrak{A}$.
  Sea $\varrho' : (\gamma^{-1}(C); E_1, \ldots, E_m)$. Es fácil ver, mediante una simple inducción, que para cada $i = 1, \ldots, m$, existe R'_i tal que $\gamma(R'_i) = R_i$, donde R'_i es un $\mathfrak{A}$-resolvente intermedio al resolver ϱ' y R_i es el correspondiente RFP-resolvente al resolver ϱ. De manera que podemos concluir que existe un R' tal que $\gamma(R') = R$, pero como R es positiva entonces R' es falsa en $\mathfrak{A}$, de manera que R' resulta ser un resolvente semántico de ϱ'. Por lo tanto $R = \gamma(R') \in \gamma(\rho_{\mathbf{S},\mathfrak{A}}(S^n_{\mathbf{S},\mathfrak{A}}(\mathbb{D})))$.

  $\subseteq$) Es totalmente análogo.

De manera que podemos concluir que $\gamma_{\mathbf{D}}(R_{\mathbf{S},\mathfrak{A}}(\mathbb{D})) = R_H(\gamma_{\mathbf{D}}(\mathbb{D}))$ ⊣

Lema 2.7 *(Completud de la resolución semántica para cláusulas cerradas). Si $\mathbb{S}$ es un conjunto de cláusulas, $\mathfrak{A}$ es una interpretación de $\mathbb{S}$ y $\mathbb{D}$ es un conjunto no satisfacible de instancias cerradas de $\mathbb{S}$, entonces $\square \in R_{\mathbf{S},\mathfrak{A}}(\mathbb{D})$.*

dem:

Como $\mathbb{D}$ es no satisfacible entonces $\gamma_{\mathbf{D}}(\mathbb{D})$ es no satisfacible, de manera que utilizando la completud de la hiperresolución para cláusulas cerradas tenemos que $\square \in R_H(\gamma_{\mathbf{D}}(\mathbb{D}))$. Por último mediante el lema 2.6 obtenemos que $\square \in \gamma_{\mathbf{D}}(R_{\mathbf{S},\mathfrak{A}}(\mathbb{D}))$, lo cual implica que $\square = \gamma^{-1}_{\mathbf{D}}(\square) \in R_{\mathbf{S},\mathfrak{A}}(\mathbb{D})$. ⊣

El siguiente lema nos muestra una propiedad de levantamiento, mediante la cual concluiremos el teorema de completud correspondiente.

Lema 2.8 *Si $\mathbb{D} \subset IC(\mathbb{S})$ entonces $R_{\mathbf{S},\mathfrak{A}}(\mathbb{S}) \leq_s R_{\mathbf{S},\mathfrak{A}}(\mathbb{D})$.*

dem:

La demostración se hace por inducción de la manera usual para los estratos correspondientes, los detalles se dejan como ejercicio, aquí sólo probaremos que $\rho_{\mathbf{S},\mathfrak{A}}(S^n_{\mathbf{S},\mathfrak{A}}(\mathbb{S})) \leq_s \rho_{\mathbf{S},\mathfrak{A}}(S^n_{\mathbf{S},\mathfrak{A}}(\mathbb{D}))$.
Tal problema se reduce a mostrar que dado cualquier conflicto semántico ϱ' :

$(C'; D'_1, \ldots, D'_m)$ de elementos de $S^n_{\mathbb{S},\mathfrak{A}}(\mathbb{D})$ existe un conflicto semántico ϱ : $(C; D_1, \ldots, D_m)$ de elementos de $S^n_{\mathbb{S},\mathfrak{A}}(\mathbb{S})$ tal que para los respectivos resolventes semánticos se cumple $R' \leq_s R$ y esto a su vez se reduce a probar la preservación del orden para los $\mathfrak{A}$-resolventes intermedios, lo cual es consecuencia inmediata del lema 1. ⊣

Por último, como corolario del trabajo anterior, obtenemos el teorema de completud para la resolución semántica.

Teorema 2.9 *(Completud de la resolución semántica).*
Si $\mathbb{S}$ *es un conjunto no satisfacible de cláusulas y* $\mathfrak{A}$ *es una interpretación de* $\mathbb{S}$, *entonces* $\square \in R_{\mathbb{S},\mathfrak{A}}(\mathbb{S})$.

dem:

Sea $\mathbb{D} \subset IC(\mathbb{S})$ un conjunto finito y no satisfacible, proporcionado mediante los teoremas de Herbrand y de Compacidad. Por el lema 3 tenemos que $\square \in R_{\mathbb{S},\mathfrak{A}}(\mathbb{D})$ y por el lema 4 debe existir una clásula $C \in R_{\mathbb{S},\mathfrak{A}}(\mathbb{S})$ tal que $C \leq_s \square$, pero esto sólo es posible si $C = \square$. Por lo tanto $\square \in R_{\mathbb{S},\mathfrak{A}}(\mathbb{S})$. ⊣

2.7.2 La estrategia del conjunto de soporte

La estrategia del conjunto de soporte ([WRC65]) es un caso especial de la resolución semántica. Se trata de una estrategia de restricción que consiste en distinguir entre las cláusulas descendientes de las hipótesis y las cláusulas obtenidas a partir de la conclusión negada. Suponiendo que las hipótesis no son contradictorias, la obtención de una contradicción debe involucrar a la conclusión. Por lo tanto, esta estrategia prohibe la generación de resolventes a partir de dos cláusulas hipótesis. De manera más general se puede distinguir cualquier subconjunto satisfacible del conjunto inicial de cláusulas y prohibir la resolución entre miembros del subconjunto distinguido; solamente las cláusulas restantes tienen "soporte". Aunque basado en [WRC65] y [CL73] el procedimiento incluye algunas ideas nuestras.

Definición 2.42 Sea $\mathbb{S}$ un conjunto de cláusulas, decimos que $\mathbb{T}$ es un *conjunto de soporte* para $\mathbb{S}$ si y sólo si $\mathbb{T} \subseteq \mathbb{S}$ y $\mathbb{S} - \mathbb{T}$ es satisfacible.

Una *cláusula con soporte* $\mathbb{T}$ es una cláusula que pertenece a $\mathbb{T}$, que es un factor de una cláusula de $\mathbb{T}$ o que es resolvente de dos cláusulas donde al menos uno de los padres tiene soporte, como lo formaliza la siguiente definición; con $Fac(\mathbb{T})$ denotamos al conjunto de factores de cláusulas de $\mathbb{T}$.

Definición 2.43 Sea $\mathbb{S}$ un conjunto de cláusulas y $\mathbb{T}$ un soporte para $\mathbb{S}$. Considérese la siguiente sucesión de conjuntos:

$$\mathbb{T}_0 = \mathbb{T} \cup Fac(\mathbb{T})$$

$$\mathbb{T}_{n+1} = \bigcup_{C,D \in \mathbb{S}} \{Res(C, D) \mid C \in \mathbb{T}_n \text{ o } D \in \mathbb{T}_n\}$$

Sea $Sop_{\mathbb{T}}(\mathbb{S}) = \bigcup_{n \in \mathbf{N}} \mathbb{T}_n$. Se dice que una cláusula C tiene $\mathbb{T}$-soporte si y sólo si $C \in Sop_{\mathbb{T}}(\mathbb{S})$.

Por último definiremos qué es una R-derivación con soporte.

Definición 2.44 Sea $\mathbb{S}$ un conjunto de cláusulas y $\mathbb{T}$ un soporte para $\mathbb{S}$. Una R-derivación $\Delta : (C_1, \ldots, C_m)$ tiene $\mathbb{T}$-soporte si y sólo si para cada $i \leq m$, si C_i se obtuvo mediante resolución entonces al menos uno de los padres C_j, C_k con $j, k < i$ tiene $\mathbb{T}$-soporte .

El siguiente lema es la base para el teorema de completud del conjunto de soporte.

Lema 2.9 *Sean $\mathbb{S}$ un conjunto de cláusulas, $\mathbb{T}$ un soporte para $\mathbb{S}$ y $\mathfrak{A}$ una interpretación de $\mathbb{S}$ que satisface a $\mathbb{S} - \mathbb{T}$.*
Si $C \in R_{\mathbb{S},\mathfrak{A}}(\mathbb{S})$ entonces existe una R-derivación Δ de C a partir de $\mathbb{S}$ que tiene $\mathbb{T}$-soporte.

dem:

Por inducción sobre n, para los estratos de $R_{\mathbb{S},\mathfrak{A}}(\mathbb{S})$.

- Base de la inducción, $n = 0$. Es obvio, tomando $\Delta = C$.
- Hipótesis de Inducción. Si $C \in S^n_{\mathbb{S},\mathfrak{A}}(\mathbb{S})$ entonces existe una R-derivación Δ de C que tiene $\mathbb{T}$-soporte.
- Paso Inductivo. Sea $C \in S^{n+1}_{\mathbb{S},\mathfrak{A}}(\mathbb{S})$. Basta analizar el caso en que $C \in \rho_{\mathbb{S},\mathfrak{A}}(S^n_{\mathbb{S},\mathfrak{A}}(\mathbb{S}))$. En tal caso tenemos que C es un resolvente semántico del conflicto $\varrho : (E; D_1, \ldots, D_k)$ donde $E, D_j \in S^n_{\mathbb{S},\mathfrak{A}}(\mathbb{S})$. La hipótesis de inducción nos proporciona R-derivaciones de E, D_j, digamos Δ_E, Δ_j a partir del conjunto $\mathbb{S}$. Tomemos la concatenación de tales derivaciones, $\Delta' = \Delta_E \star \Delta_1 \star \ldots \star \Delta_k$. Por otra parte consideramos los $\mathfrak{A}$-resolventes internos del conflicto ϱ y con ellos obtenemos la derivación $\Delta'' : (R_1, \ldots, R_k)$, pero como $R_k = C$, entonces la derivación buscada será $\Delta = \Delta' \star \Delta''$.

⊣

Como corolario del lema anterior, obtenemos el teorema de completud para el conjunto de soporte, tal y como se enuncia en [CL73].

Teorema 2.10 *(Completud de la Estrategia del Conjunto de Soporte).*
Si $\mathbb{S}$ es un conjunto no satisfacible de cláusulas y $\mathbb{T}$ es un soporte para $\mathbb{S}$, entonces existe una R-refutación con $\mathbb{T}$-soporte de $\mathbb{S}$.

dem:

Sea $\mathfrak{A}$ una interpretación de $\mathbb{S}$ que satisface a $\mathbb{S} - \mathbb{T}$.
Por el teorema de completud para la resolución semántica, tenemos que $\square \in R_{\mathbb{S},\mathfrak{A}}(\mathbb{S})$, de manera que, utilizando el lema anterior, tenemos una R-derivación con $\mathbb{T}$-soporte a partir del conjunto $\mathbb{S}$ de $\square$, es decir una R-refutación de $\mathbb{S}$, con $\mathbb{T}$-soporte. ⊣

En este capítulo hemos presentado diversas reglas de inferencia, basadas en la regla de resolución binaria, que generan procesos de decisión que están implementados en OTTER. Dichos procesos de decisión funcionan correctamente debido a los teoremas de completud presentados aquí.
Además presentamos una introducción general a los operadores de refinamiento, la cual sirve de base para estudios posteriores sobre nuevos procesos de decisión o construcción de modelos. Sin embargo todos los resultados anteriores no involucran al predicado de igualdad, al menos no explicitamente, aunque podrían servir en casos particulares al considerar la igualdad como un predicado más sin explotar sus propiedades características.
El punto de estudio del siguiente capítulo es precisamente la igualdad mediante la regla de paramodulación y los sistemas de reescritura de términos.

Capítulo 3

La Igualdad

Hasta este momento hemos trabajado con símbolos de predicado arbitrarios. Sin embargo, existe un símbolo de predicado distinguido, el símbolo $=$, que representa a la relación de igualdad. Es ésta una relación tan importante que su tratamiento dentro del razonamiento automático merece atención especial. En este capítulo analizaremos la regla de paramodulación desarrollada originalmente por Larry Wos y George Robinson en 1969-1970 para manejar la igualdad [RW69, WR70] y hablaremos del concepto de demodulación también desarrollado por Wos y sus colegas [WRC67].
Por último presentaremos una breve introducción al tema de sistemas de reescritura de términos, especialmente al método de compleción de Knuth-Bendix, presentado originalmente en [KB70]. Cabe mencionar que los procesos de paramodulación y demodulación están implementados en OTTER, mientras que el proceso de Knuth-Bendix no se implementa directamente, pero se modela con ayuda de otros procesos y opciones que ofrece el programa.

Empezamos recordando cuáles son los axiomas de igualdad.

Definición 3.1 Sea Σ una signatura. El conjunto de *axiomas de igualdad para* Σ, *denotado* $\mathbb{E}_\Sigma$ *o simplemente* $\mathbb{E}$ *es el conjunto de las siguientes cláusulas:*

1. Axioma de Reflexividad, $x = x$.
2. Axioma de Simetría, $x \neq y \vee y = x$.
3. Axioma de Transitividad, $x \neq y \vee y \neq z \vee x = z$.
4. Para cada $f \in \Sigma$, $x_1 \neq y_1 \vee \ldots \vee x_n \neq y_n \vee f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$.
5. Para cada $P \in \Sigma$, $x_1 \neq y_1 \vee \ldots \vee x_n \neq y_n \vee \neg P(x_1, \ldots, x_n) \vee P(y_1, \ldots, y_n)$.

La siguiente notación será de gran utilidad en adelante, Si L es una literal entonces $L[t]$ indica una presencia del término t en L. Además si se dice que se reemplaza $L[r]$ por $L[s]$ en alguna regla de inferencia, la notación llevará implicito que el reemplazo se hizo en la misma posición. Lo mismo sucede con la notación $t[r]$ para términos y $C[r]$ para cláusulas.

3.1 Demodulación

Demodulación es el proceso de reescribir expresiones mediante la aplicación automática de cláusulas unitarias de igualdad diseñadas para este propósito (Cf. [WRC67]). Una igualdad de tal tipo se llama un *demodulador*. Típicamente, todos los términos de las cláusulas recién generadas por un programa de razonamiento automático se examinan para una posible demodulación. Un término es demodulado si existe un demodulador tal que uno de sus argumentos pueda unificarse con el término sin reemplazar ninguna de las variables del término. Una convención común es utilizar únicamente el argumento de la izquierda del demodulador para unificación con el término en consideración. El término demodulado se obtiene reemplazando el término por el segundo argumento del demodulador después de haber aplicado el unificador. Finalmente se sustituye la claúsula original por la cláusula demodulada. Cuando un programa de razonamiento automático encuentra un nuevo demodulador, dependiendo de las instrucciones que se le hayan dado, todas las cláusulas retenidas previamente se examinan para una posible demodulación con el nuevo demodulador. Este proceso se conoce como *demodulación hacia atrás.*
El proceso de demodulación expresado como regla de inferencia es el siguiente:

$$\frac{\begin{array}{c} s = t \\ C[r] \end{array}}{C[t\sigma]}$$

donde σ es un unificador de $\{s, r\}$. $s = t$ es el demodulador y $C[t\sigma]$ es la cláusula demodulada que se obtuvo reemplazando la presencia distinguida de r en C por $t\sigma$. El proceso no funciona como regla de inferencia sino como regla de reescritura, pues la cláusula original se elimina sustituyéndose por la cláusula demodulada.

Ejemplo 3.1 A partir de la cláusula $C : P(f(x,c),a) \vee Q(x,w)$ y del demodulador $f(y,w) = g(w,y)$ obtenemos la cláusula demodulada $C' : P(g(c,x),a) \vee Q(x,w)$ mediante el unificador $\sigma = \{y/x, w/c\}$. A partir de este momento se elimina la cláusula C a favor de C'.

Este proceso, si bien de gran utilidad, no es suficiente para simplificar igualdades, pues puede darse el caso en que dos términos se demodulen a términos diferentes que ya no puedan ser demodulados a un mismo término.

3.2 Paramodulación

La última regla de inferencia que analizaremos es la regla de *paramodulación*. Paramodulación es una generalización de la regla de substitución de iguales en el sentido de que combina en un solo paso la busqueda de una sustitución que permite un reemplazo de igualdad con el reemplazo en si mismo. Mientras las reglas de inferencia que hemos visto anteriormente están en el nivel de literales,

esta regla está orientada hacia el nivel de los términos, situación que resulta clara, puesto que las igualdades se dan siempre entre términos.
La regla de paramodulación es la siguiente:

$$\frac{u = v \vee M_2 \vee \ldots \vee M_n \qquad L_1[s] \vee L_2 \vee \ldots \vee L_m}{(L_1[v] \vee L_2 \vee \ldots \vee L_m \vee M_2 \vee \ldots \vee M_n)\sigma}$$

donde la notación $L[s]$ distingue una presencia del término s en la literal L y σ es un umg de $\{s, u\}$. Dentro de las premisas la clásula con la igualdad se llama *cláusula origen* o "desde" y la otra cláusula se llama *cláusula destino* o "hacia"; la conclusión se llama *paramodulante*. Obsérvese que en la paramodulación se está haciendo una demodulación a una de las literales de las cláusulas padres. Veamos un par de ejemplos

Ejemplo 3.2

$$\frac{a = b \vee R(b) \qquad P(a) \vee Q(b)}{P(b) \vee Q(b) \vee R(b)}$$

en este caso el término distinguido es a y el unficador es $\sigma = \varnothing$.

Ejemplo 3.3

$$\frac{f(g(b)) = a \vee R(g(c)) \qquad P(g(f(x))) \vee Q(x)}{P(g(a)) \vee Q(g(b)) \vee R(g(c))}$$

aquí el término distinguido es $f(x)$ y el unificador es $\sigma = \{x/g(b)\}$

En esta sección probaremos que la combinación de RFP-resolución y paramodulación es completa siguiendo el método de modificación de Brand [Bra75]. Todo el tiempo trabajaremos en la lógica de predicados sin igualdad, de esta manera, estamos justificando el uso de la igualdad sin usar a la igualdad misma.

3.2.1 S-Interpretaciones y E-Modelos

En este apartado probaremos como se pueden garantizar las propiedades de substitución de la igualdad evitando el mayor número posible de axiomas de igualdad. Como necesitamos asegurar que $=$ es una relación de equivalencia, incluiremos los axiomas $x = x$ y el siguiente axioma que es una forma débil de transitividad.

$$A \equiv x \neq y \vee x \neq z \vee y = z,^1$$

de este último axioma prescindiremos posteriormente.

Lo primero es ordenar todas las literales cerradas mediante una numeración de Gödel.

[1] Durante toda la sección con $\equiv$ denotamos a la igualdad en el metalenguaje.

Definición 3.2 Sea Σ una signatura. Asociamos a cada símbolo $\circ \in \Sigma$ un único número natural impar denotado $G(\circ)$. Definimos el *número de Gödel* del término cerrado $t \equiv f(t_1, \ldots, t_n)$ recursivamente como sigue:

$$G(t) \equiv 2^{G(f)} \cdot \prod_{i=1}^{n} Pr_{i+1}^{G(t_i)}$$

donde Pr_i es el i-ésimo número primo.

Proposición 3.1 *Los números de Gödel cumplen con las siguientes propiedades:*

1. *Si t es un subtérmino de s, entonces $G(t) \leq G(s)$ y la igualdad se da syss $s \equiv t$.*
2. *Si $G(t) < G(s)$, s es un subtérmino de r, y r' se obtiene de r reemplazando una presencia de s por t, entonces $G(r') < G(r)$.*

dem:

Por inducción sobre los términos cerrados. ⊣

La noción de número de Gödel se extiende a literales y cláusulas como sigue.

Definición 3.3 Sea L una literal positiva. Definimos el *número de Gödel* de L como

$$G(L) = \begin{cases} 2^{G(P)} \cdot \prod_{i=1}^{n} Pr_{i+1}^{G(t_i)} & \text{si } L \equiv P(t_1, \ldots, t_n) \\ G(s) & \text{si } L \equiv s = t \end{cases}$$

En el caso de que L sea una literal negativa definimos $G(L) = G(L^c)$.

Proposición 3.2 *Si L es una literal en la que figura s entonces*

$$G(s = t) < G(L[s]),$$

exceptuando posiblemente los casos $L \equiv s = r$ o $L \equiv s \neq r$.[2]

dem:

Resulta claro a partir de la definición de G. ⊣

Definición 3.4 Sean C una cláusula cerrada y $t_1, \ldots, t_n$ todos los términos que figuran en C a nivel de argumento. Definimos el número de Gödel de C como

$$G(C) = \sum_{i=1}^{n} G(t_i)$$

[2] La desigualdad tampoco es cierta si $G(p) < G(s)$ y $L \equiv p = s$ o $L \equiv p \neq s$ pero este caso no será permitido.

Proposición 3.3 *Si $G(s) < G(t)$ y la cláusula C' se obtuvo a partir de C reemplazando algunas presencias de s por t entonces $G(C) < G(C')$.*

dem:

Por inducción sobre los términos cerrados. ⊣

Para evitar exceso de símbolos, algunas veces abusaremos de la notación escribiendo $C < C'$ en lugar de $G(C) < G(C')$.

Definición 3.5 Una *interpretación simétrica o S-interpretación* es una asignación de valores (verdadero o falso) a cada literal cerrada, la cual interpreta a la igualdad de manera simétrica, es decir, $s = t$ y $t = s$ tienen el mismo valor de verdad.

Generalmente denotamos a una S-interpretación como el conjunto de cláusulas que son verdaderas en ella. Para facilitar el trabajo, en el caso de la igualdad, sólo utilizaremos igualdades orientadas, de manera que el término más complejo sea el del lado izquierdo. Por ejemplo una S-interpretación puede contener a la igualdad $f(b) = b$ pero no a $b = f(b)$. Esto se formaliza en la siguiente definición.

Definición 3.6 Una literal L pertenece a la S-interpretación M syss L es verdadera en M y no es de la forma $s = t$ o $s \neq t$, donde $G(s) < G(t)$.

Si tenemos una S-interpretación *parcial*, es decir, un subconjunto (no necesariamente propio) de una S-interpretación, a las literales de ésta las ordenamos según sus números de Gödel. Las literales con igual número de Gödel son las igualdades o sus negaciones y se ordenan según el número del término de la derecha. Obsérvese que si la literal $s = t$ pertenece a una S-interpretación entonces s no es un subtérmino propio de t.

Definición 3.7 Una S-interpretación M es un *S-modelo* para un conjunto de cláusulas $\mathbb{S}$ *syss* M es un modelo de $IC(\mathbb{S})$.

Resulta adecuado observar que $\mathbb{S}$ tiene un S-modelo syss $\mathbb{S} \cup \{x \neq y \vee y = x\}$ tiene un modelo.

Definición 3.8 Sea M una S-interpretación parcial. Una *E-inconsistencia c.r.a.*[3] $s = t$ en M es un par de literales $(L[s], \neg L[t])$ que son verdaderas en M. Una *E-inconsistencia* en M es una terna $(s = t, L[s], \neg L[t])$ tal que $s = t$ es verdadera en M y $(L[s], \neg L[t])$ es una E-inconsistencia c.r.a. $s = t$. M es *E-consistente c.r.a.* $s = t$ si no contiene E-inconsistencias c.r.a. $s = t$. M es *E-consistente* si no tiene E-inconsistencias.

Las literales de una S-interpretación M estan ordenadas de manera que si M es modelo de $\{x \neq y \vee x \neq z \vee y = z, x = x\}$ y $(s = t, L[s], \neg L[t])$ es una E-inconsistencia en M entonces $L[s]$ es mayor que $s = t$ y $\neg L[t]$.

[3] c.r.a. abrevia "con respecto a".

Definición 3.9 Una S-interpretación M es un *E-modelo* para $\mathbb{S}$ syss M es un S-modelo para $\mathbb{S} \cup \{x = x\}$ y M es E-consistente.

Proposición 3.4 *M es un E-modelo para $\mathbb{S}$ syss M es un modelo para $\mathbb{S} \cup \mathbb{E}$.*

dem:

$\Leftarrow$) Como M es un modelo de los axiomas de igualdad, basta ver que M es E-consistente, pero esto es inmediato, pues la existencia de una E-inconsistencia contradice a los axiomas de igualdad.
$\Rightarrow$) Supongamos lo contrario. Tenemos tres casos:

1. M no satisface al axioma

$$x \neq y \vee \neg P(x_1, \ldots, x, \ldots, x_n) \vee P(x_1, \ldots, y, \ldots, x_n),$$

entonces existen términos cerrados $r_1, \ldots, r_n, s, t$ tales que las tres literales $s = t, P(r_1, \ldots, s, \ldots, r_n), \neg P(r_1, \ldots, t, \ldots, r_n)$ son verdaderas en M, lo cual contradice la E-consistencia de M.

2. M no satisface al axioma de transitividad

$$x \neq y \vee y \neq z \vee x = z$$

de manera que existen términos cerrados t_1, t_2, t_3 tales que las literales $t_1 = t_2, t_2 = t_3, t_1 \neq t_3$ son verdaderas en M. En tal caso, dichas literales constituyen una E-inconsistencia en M, lo cual es absurdo.

3. M no satisface al axioma

$$x \neq y \vee f(x_1, \ldots, x, \ldots, x_n) = f(x_1, \ldots, y, \ldots, x_n),$$

entonces hay términos cerrados $r_1, \ldots, r_n, s, t$ tales que las dos literales $s = t, f(r_1, \ldots, s, \ldots, r_n) \neq f(r_1, \ldots, t, \ldots, r_n)$ son verdaderas en M y como $f(r_1, \ldots, s, \ldots, r_n) = f(r_1, \ldots, s, \ldots, r_n)$ es verdadera en M entonces M es E-inconsistente c.r.a. $s = t$, lo cual contradice la E-consistencia de M.

⊣

Definición 3.10 Sean M una S-interpretación, s, t términos tales que $G(s) > G(t)$. Para cada literal $L \in M$ definimos la S-interpretación parcial M_L recursivamente como sigue:
Supóngase que M_K está definida para cada $K \in M$ tal que $K < L$. Sea $M'_L = \bigcup_{K<L} M_K$. Hay dos casos:

1. Si s no figura en L definimos $M_L = M'_L \cup \{L\}$.

2. Supongamos que s figura en L. Sea L' la literal obtenida a partir de L reemplazando una presencia arbitraria de s por t.[4] Por la proposición 3.3 tenemos que $L' < L$. Definimos

$$M_L = \begin{cases} M'_L \cup \{L\} & \text{Si } M'_L \models L' \\ M'_L \cup \{\neg L\} & \text{en otro caso.} \end{cases}$$

Proposición 3.5 *La S-interpretación parcial M_L tiene las siguientes propiedades:*

1. *Para cada $K \in M$, si $K \leq L$ entonces $K \in M_L$ o $\neg K \in M_L$ y sólo una. Además estas son las únicas literales que pertenecen a M_L.*
2. *M_L es E-consistente c.r.a. $s = t$.*
3. *Si $K \leq L$ entonces $M_K \subseteq M_L$.*

dem:

Por <-Inducción sobre L.

a) s no figura en L. Las propiedades 1 y 3 son claras. La propiedad 2 se cumple puesto que si hay una E-inconsistencia c.r.a. $s = t$, por HI, en ella tiene que participar L. Así que L tendría que ser de la forma $L[t]$ donde $\neg L[s]$ también es verdadera en M, lo cual no es posible pues, por la proposición 3.3 se tiene $L[t] < \neg L[s]$ lo cual no es posible según lo observado al terminar la definición 3.8.

b) s figura en L. Nuevamente las propiedades 1 y 3 resultan claras. La propiedad 2 se cumple por construcción de M_L.

⊣

Definición 3.11 Sean M una S-interpretación y s, t términos tales que $G(s) > G(t)$. Definimos la S-interpretación $M[s/t]$ como

$$M[s/t] = \bigcup_{L \in M} M_L$$

Proposición 3.6 *$M[s/t]$ cumple lo siguiente:*

a) Si s no figura en L entonces $M \models L$ syss $M[s/t] \models L$.

b) $M[s/t]$ es E-consistente c.r.a. $s = t$

c) $s = t \in M[s/t]$ sólo si $t = t \in M$.

[4] La definición es independiente de la elección de una presencia de s pues, como veremos adelante, M_L será E-consistente c.r.a. $s = t$.

dem:

Esta demostración no figura en la literatura sobre el tema.

a) $\Rightarrow$) Supongamos que $M \models L$ y s no figura en L. Entonces $L \in M_L = M'_L \cup \{L\} \subseteq M[s/t]$. Por lo tanto $M[s/t] \models L$.
$\Leftarrow$) Si $L \notin M$ entonces $\neg L \in M$, así que por lo anterior tenemos $\neg L \in M[s/t]$, es decir $M[s/t] \not\models L$.

b) Si existiera L tal que $M[s/t] \models \{L[s], \neg L[t], s = t\}$. Tendríamos que $M_L = M'_L \cup \{\neg L\}$, por lo tanto $\neg L \in M[s/t]$, lo cual es absurdo, pues $M[s/t] \models L$.

c) Si $t = t \notin M$ entonces, por a), $t = t \notin M[s/t]$, es decir, $t \neq t \in M[s/t]$. Si además suponemos que $s = t \in M[s/t]$ tendríamos que $(s = t, t \neq t, s = t)$ es una E-inconsistencia, lo cual contradice a b).

$\dashv$

Definición 3.12 Un *término no variable* es un término que no es una variable, es decir, es una constante o un término compuesto.

Definición 3.13 Una cláusula C está en *E-forma* syss todo término no variable que figura en C lo hace únicamente como argumento de $=$ ó de $\neq$.

Ejemplo 3.4 La cláusula $C \equiv f(c) = x \vee g(x, y) = w \vee y = z$ no está en E-forma pues el término no variable c no figura como argumento de igualdad sino como argumento de un símbolo de función.

Más adelante veremos como construir, dada una cláusula C, una cláusula C' en E-forma tal que $C \sim_{sat} C'$.

Lema 3.1 *Sea M un S-modelo para un conjunto de cláusulas en E-forma $\mathbb{S}$ que contiene a los axiomas $A \equiv x \neq y \vee x \neq z \vee y = z$ y $x = x$. Si $s = t \in M$ entonces $M[s/t]$ es un S-modelo para $\mathbb{S}$.*

dem:

Supongamos lo contrario. En tal caso existen $C_0 \in \mathbb{S}$ y $C \in IC(\mathbb{S})$ tales que C es una instancia de C_0 con número de Gödel mínimo entre las cláusulas que cumplen $M[s/t] \not\models C$. Hay tres posibilidades:

- s no figura en C. Entonces $M \not\models C$, por la proposición 3.6a, lo cual es absurdo, pues M es un S-modelo para $\mathbb{S}$.

- Hay una presencia de s en C que no es argumento de igualdad. Sea σ tal que $C = C_0\sigma$. Como C_0 está en E-forma σ debe asignar un término $r[s]$ a una variable x de C_0. Sea σ' tal que $x\sigma' = r[t]$ y $y\sigma' = y\sigma$ si $x \neq y$. Entonces se tiene $G(C_0\sigma') < G(C)$ de donde $M[s/t] \models C_0\sigma'$, por

la minimalidad de C. El valor de verdad de todas las literales de C es el mismo que el de las literales de $C_0\sigma'$, pues de lo contrario habría una E-inconsistencia c.r.a. $s = t$ en $M[s/t]$. Por lo tanto $M[s/t] \models C$ lo cual es absurdo.

- Todas las presencias de s en C son argumentos de igualdad.
  Basta observar que el valor de verdad de las literales $s = r, r = s, s \neq r, r \neq s$ en $M[s/t]$ es el mismo que en M. Por ejemplo, si $M \models s = r$, para que $M[s/t] \not\models s = r$ se tendría que cumplir $M \models t \neq r$ (de lo contrario $M \models t = r$ y por 3.6a, $M[s/t] \models t = r$, obteniéndose una E-inconsistencia c.r.a. $s = t$, lo cual es absurdo por 3.6b).
  Así que tendríamos $M \models s = r \wedge t \neq r$, pero esto contradice claramente a los axiomas A y $x = x$ debido a que $s = t \in M$. Por lo tanto $M[s/t] \models C$ lo cual contradice a la elección de C.

⊣

Teorema 3.1 *Sea* $\mathbb{S}$ *un conjunto satisfacible de cláusulas en E-forma tal que* $A, x = x \in \mathbb{S}$. *Entonces* $\mathbb{S}$ *tiene un E-modelo.*

dem:

Tenemos que ver que $\mathbb{S} \cup \{x = x\} = \mathbb{S}$ tiene un S-modelo E-consistente, es decir, debemos mostrar que existe una S-interpretación E-consistente que es modelo de $IC(\mathbb{S})$. Como $A, x = x \models x \neq y \vee y = x$ entonces cualquier interpretación que sea modelo de $IC(\mathbb{S})$ es simétrica. Por lo tanto basta ver que $IC(\mathbb{S})$ tiene un modelo E-consistente.
Como $\mathbb{S}$ es satisfacible tiene un modelo y, por el teorema de Herbrand 1.3, $IC(\mathbb{S})$ tiene un modelo. Así que basta ver que hay un modelo E-consistente. Para esto construiremos modelos M de $IC(\mathbb{S})$ tal que dado cualquier $n \in \mathbb{N}$ el conjunto M_n de las primeras n literales de M resulta E-consistente.
Sea M un S-modelo para $\mathbb{S}$ y n máxima tal que M_n es E-consistente, si no hay tal n entonces M es consistente y terminamos. En tal situación existen términos s, t y una literal L tales que $L[s]$ es la (n+1)-ésima literal de M y $(s = t, L[s], \neg L[t])$ es una E-inconsistencia en M. A partir de M vamos a construir un modelo M^* de $\mathbb{S}$ tal que M^*_{n+1} sea E-consistente. Sea $M^* = M[s/t]$, por el lema 3.1 este es un modelo de $\mathbb{S}$. Como M_n es E-consistente tenemos que $M^*_n = M_n$, así que para mostrar la E-consistencia de M^*_{n+1} basta ver que no hay inconsistencias que involucren a la $(n+1)$-ésima literal de M^* que es $\neg L[s]$. Supongamos lo contrario, esto obliga a que exista una E-inconsistencia, digamos c.r.a. $p = q \in M_n$ $(G(q) < G(p))$; por lo que p debe figurar en $\neg L[s]$. Tenemos *tres casos:*

- p es subtérmino de s. Entonces $s \equiv s[p]$.
  Dado que $s[p] = t \in M_n$ y $s[q] = t \in M$ entonces, como $G(s[q]) < G(s[p])$ tenemos $s[q] = t \in M_n$ o $t = s[q] \in M_n$ y además $L[s[p]] \in M_n$. Supongamos spg que $s[q] = t \in M_n$, esto lleva a la E-inconsistencia $(s[q] = t, \neg L[t], L[s[q]])$ en M_n contradiciendo la E-consistencia de M_n.

- s es un subtérmino de p. Entonces $p \equiv p[s]$ y $L[s] \equiv L[p[s]]$.
  De manera análoga al caso anterior podemos suponer que $p[t] = q \in M_n$. Así tenemos la E-inconsistencia $(p[t] = q, \neg L[p[t]], L[q])$ en M_n, lo cual resulta absurdo.

- s y p son términos ajenos. Entonces $L[s] \equiv L[s,p]$, así que $\neg L[t,p], L[s,q] \in M_n$. Si $M \models L[t,q]$, $(p = q, \neg L[t,p], L[t,q])$ es una E-inconsistencia en M_n. En caso contrario $(s = t, L[s,q], \neg L[t,q])$ es una E-inconsistencia en M_n. Nuevamente obtuvimos una contradicción.

⊣

Para terminar la sección mostramos como eliminar el axioma A.

Definición 3.14 Una cláusula C está en *forma transitiva* o *T-forma* si cada igualdad que figure en C es de la forma $s = w$ donde w es una variable, C contiene también una literal $t \neq w$ y estas son las únicas presencias de w en C.

Una cláusula se transforma en T-forma si cada una de sus igualdades $s = t$ se sustituye por $s = w \vee t \neq w$, donde w es una variable que no figuraba en la cláusula.

Definición 3.15 Un conjunto de cláusulas $\mathbb{S}$ está en *forma simétrica-transitiva* o *ST-forma* si cada cláusula de $\mathbb{S}$ está en T-forma y si $(t \neq w \vee s = w) \vee C') \in \mathbb{S}$ entonces $(s \neq w \vee t = w) \vee C') \in \mathbb{S}$.

En una cláusula en ST-forma tenemos que se consideran ambos lados de una igualdad $s = t$ y $t = s$ sustituyéndolos por sus T-formas.

Proposición 3.7 *Sea* $\mathbb{S}$ *un conjunto de cláusulas en ST-forma. Entonces*

$$\mathbb{S} \cup \{x = x\} \sim_{sat} \mathbb{S} \cup \{A, x = x\}$$

dem:

Véase [Bra75]. ⊣

3.2.2 El Método de modificación

La noción fundamental para el método de modificación es la de *socio*. La idea es la siguiente: "sacar" un término t de una literal $P(t)$ y hacerlo su socio mediante una variable. Por ejemplo, la literal $P(f(a))$ se reemplaza por $f(a) \neq w \vee P(w)$, donde w es una variable nueva. A $f(a) \neq w$ se le llama *socio* de $P(w)$. A cada literal se le asocia un número natural llamado el *nivel*, en el ejemplo anterior el nivel de Pw es 0 y el nivel de $f(a) \neq w$ es 1. La literal $f(a) \neq w$ se sigue procesando sacando el término a obteniéndose la cláusula $a \neq u \vee f(u) \neq w$, $a \neq u$ es socio de $f(u) \neq w$ y su nivel es 2. Una literal se puede ver como un árbol, donde los nodos son los socios y en la raíz figura el predicado original.

Definición 3.16 Una *cláusula aplanada* C es un bosque de literales tal que cada literal L que no es raíz tiene la forma $s \neq w$, donde la variable w tiene exactamente otra presencia en C en el predecesor inmediato K de L. En tal situación L es *socio* de K. Si K es de la forma $t = w$ entonces L es un *T-socio* de K y (L, K) es un par de T-socios.

Definición 3.17 A toda literal de una cláusula aplanada se le asocia un *nivel* como sigue: las raices tienen nivel 0 y los socios de K tienen el nivel de K más uno.

Definición 3.18 La *E-modificación* $EMod(C)$ de una cláusula C se define recursivamente como sigue:

- Si C está en E-forma entonces $EMod(C) \equiv C$.
- Sea s un término no variable que figura en C pero no como argumento de $=$ y sea C' obtenida a partir de C reemplazando $L[s]$ por $s \neq w \vee L[w]$ donde w es una variable nueva. Entonces $EMod(C) \equiv EMod(C')$.

Ejemplo 3.5 Sean $D \equiv Q(ha, gb, x) \vee ga = hb$, $C \equiv Pf(a, gx) \vee f(x, y) = g(a)$. Construimos detalladamente sus E-modificaciones:

$$
\begin{aligned}
EMod(D) &\equiv EMod(D') \\
D' &\equiv ha \neq u \vee Q(u, gb, x) \vee ga = hb \\
EMod(D') &\equiv EMod(D'') \\
D'' &\equiv a \neq v \vee hv \neq u \vee Q(u, gb, x) \vee ga = hb \\
EMod(D'') &\equiv EMod(D''') \\
D''' &\equiv a \neq v \vee hv \neq u \vee gb \neq w \vee Q(u, w, x) \vee ga = hb \\
EMod(D''') &\equiv EMod(D^{iv}) \\
D^{iv} &\equiv a \neq v \vee hv \neq u \vee b \neq y \vee gy \neq w \vee Q(u, w, x) \vee ga = hb \\
EMod(D^{iv}) &\equiv EMod(D^{v}) \\
D^{v} &\equiv a \neq v \vee \ldots \vee Q(u, w, x) \vee a \neq z \vee gz = hb \\
EMod(D^{v}) &\equiv EMod(D^{vi}) \\
D^{vi} &\equiv a \neq v \vee \ldots \vee Q(u, w, x) \vee a \neq z \vee b \neq x_1 \vee gz = hx_1 \\
EMod(D^{vi}) &\equiv D^{vi}
\end{aligned}
$$

Por lo tanto $EMod(D) \equiv D^{vi}$.

Para C tenemos lo siguiente:

$$\begin{aligned} EMod(C) &\equiv EMod(C') \\ C' &\equiv f(a,gx) \neq w \vee Pw \vee f(x,y) = ga \\ EMod(C') &\equiv EMod(C'') \\ C'' &\equiv a \neq u \vee f(u,gx) \neq w \vee Pw \vee f(x,y) = ga \\ EMod(C'') &\equiv EMod(C''') \\ C''' &\equiv a \neq u \vee gx \neq v \vee f(u,v) \neq w \vee Pw \vee f(x,y) = ga \\ EMod(C''') &\equiv EMod(C^{iv}) \\ C^{iv} &\equiv a \neq u \vee gx \neq v \vee f(u,v) \neq w \vee Pw \vee a \neq z \vee f(x,y) = gz \\ EMod(C^{iv}) &\equiv C^{iv} \end{aligned}$$

Por lo tanto $EMod(C) \equiv C^{iv}$
Es claro que toda E-modificación está en E-forma.

Definición 3.19 Sea C una cláusula sin T-socios. La *T-modificación* $TMod(C)$ de C se obtiene reemplazando cada igualdad $s = t$ de C por el nuevo par de T-socios $t \neq w \vee s = w$. Donde w es una variable nueva llamada la nueva T-variable.

Obsérvese que los T-socios tienen nivel 1.

Ejemplo 3.6 Las T-modificaciones de las cláusulas del ejemplo 3.5 son:

$$\begin{aligned} TMod(C) &\equiv Pf(a,gx) \vee ga \neq w \vee f(x,y) = w \\ TMod(D) &\equiv Q(ha,gb,x) \vee hb \neq v \vee ga = v \end{aligned}$$

Toda T-modificación esta en T-forma.

Definición 3.20 La *ST-modificación* $STMod(\mathbb{S})$ de un conjunto de cláusulas $\mathbb{S}$ se obtiene como sigue: Sea $\mathbb{S}'$ el mínimo conjunto que contiene a $\mathbb{S}$ y que cumple que si $s = t \vee C \in \mathbb{S}'$ entonces $t = s \vee C \in \mathbb{S}'$. Entonces $STMod(\mathbb{S}) \equiv TMod(\mathbb{S}')$

Ejemplo 3.7 Siguiendo con las cláusulas del ejemplo 3.5, sea $\mathbb{S} = \{C, D\}$. La ST-Modificación de $\mathbb{S}$ es:

$$\begin{aligned} STMod(\mathbb{S}) \;=\; & \{Pf(a,gx) \vee ga \neq w \vee f(x,y) = w, \\ & Pf(a,gx) \vee f(x,y) \neq w \vee ga = w, \\ & Q(ha,gb,x) \vee hb \neq v \vee ga = v, \\ & Q(ha,gb,x) \vee ga \neq v \vee hb = v\} \end{aligned}$$

Definición 3.21 Sea $\mathbb{S}$ un conjunto de cláusulas y $\mathbb{S}' = EMod(\mathbb{S})$. Entonces la *STE-Modificación* $STEMod(\mathbb{S})$ de $\mathbb{S}$ es $STMod(\mathbb{S}')$. Es decir, $STEMod(\mathbb{S}) = STMod(EMod(\mathbb{S}))$.

Ejemplo 3.8 Sea $\mathbb{S}$ como en el ejemplo 3.7, donde

$$EMod(\mathbb{S}) = \{C \equiv C' \vee f(x,y) = gz, D \equiv D' \vee gz = hx_1\}$$

entonces

$$\begin{aligned} STEMod(\mathbb{S}) \;=\; & \{C' \vee gz \neq y_1 \vee f(x,y) = y_1, \\ & D' \vee hx_1 \neq y_2 \vee gz = y_2, \\ & C' \vee f(x,y) \neq y_3 \vee gz = y_3, \\ & D' \vee gz \neq y_4 \vee hx_1 = y_4\} \end{aligned}$$

Las diferentes modificaciones guardan una estrecha relación con los axiomas de igualdad y la resolución, como se ve en la siguiente proposición que demostramos a partir de una observación en [Bra75].

Proposición 3.8 *Sea C una cláusula. Cada paso realizado para producir cualquier modificación de C se puede hacer resolviendo C con algún axioma de igualdad.*

dem:

Vamos a analizar cada modificación.

- E-Modificación. Para producir $EMod(C)$ se sustituye cada literal $L[s]$, donde s figura como argumento de función o predicado, por $s \neq w \vee L[w]$. Digamos que $C \equiv L[s] \vee C'$ entonces la siguiente aplicación de resolución binaria a C y al axioma de igualdad $x \neq y \vee \neg L[x] \vee L[y]$ realiza el reemplazo deseado mediante el umg $\sigma = \{x/s, y/w\}$.

$$\frac{x \neq y \vee \neg L[x] \vee L[y] \qquad L[s] \vee C'}{s \neq w \vee L[w] \vee C'}$$

- T-Modificación. Para obtener $TMod(C)$ se reemplaza cada igualdad $s = t$ por $t \neq w \vee s = w$. Si $C \equiv C' \vee s = t$ entonces resolviendo C con el axioma de igualdad $x \neq y \vee y \neq w \vee x = w$ mediante el umg $\sigma = \{x/s, y/t\}$ obtenemos el reemplazo deseado.

$$\frac{C' \vee s = t \quad x \neq y \vee y \neq w \vee x = w}{C' \vee t \neq w \vee s = w}$$

- ST-Modificación. Sea $C \equiv s = t \vee C'$. Para obtener $STMod(C)$, debemos obtener antes la cláusula $t = s \vee C'$ y despues obtener las T-modificaciones de ambas. Como el caso de T-modificaciones ya lo analizamos, basta ver como obtener la cláusula con la igualdad invertida.
  Tal cláusula es el resolvente de C con el axioma de igualdad $x \neq y \vee y = x$, mediante $\sigma = \{x/s, y/t\}$.

$$\frac{x \neq y \vee y = x \quad s = t \vee C'}{t = s \vee C'}$$

- STE-Modificación. Resulta obvio de los casos anteriores.

⊣

Además podemos, a partir de una modificación, recuperar la cláusula original resolviendo con $x = x$, como lo asegura la siguiente proposición también demostrada por nosotros a partir de una observación en [Bra75].

Proposición 3.9 *Si D es $EMod(C)$ o $TMod(C)$ para alguna cláusula C y D' es la cláusula que resulta al resolver todos los socios de D con el axioma de igualdad $x = x$ entonces D' es igual a C.*

dem:

Basta analizar que sucede al resolver los socios.
En una E-modificación, el reemplazo de $L[s]$ por $s \neq w \vee L[w]$ se puede invertir resolviendo con $x = x$ mediante $\sigma = \{x/s, w/s\}$.

$$\frac{x = x \quad s \neq w \vee L[w]}{L[s]}$$

Análogamente, para revertir la sustitución de $s = t$ por el par de T-socios $t \neq w \vee s = w$, se resuelve este par con $x = x$ mediante $\sigma = \{x/t, w/t\}$.

$$\frac{x = x \quad t \neq w \vee s = w}{s = t}$$

⊣

Con el siguiente teorema conseguimos un método para decidir la satisfacibilidad de un conjunto de cláusulas con igualdad. Con la ayuda de la proposición anterior mejoramos la demostración original de [Bra75].

Teorema 3.2 *Sea* $\mathbb{S}$ *un conjunto de cláusulas.*
$\mathbb{S}$ *tiene un E-modelo syss* $STEMod(\mathbb{S}) \cup \{x = x\}$ *tiene un modelo.*

dem:

$\Rightarrow$) Sea M un E-modelo para $\mathbb{S}$, por la proposición 3.4 M es un modelo para $\mathbb{S} \cup \mathbb{E}$, en particular $M \models x = x$. Así, como la resolución es correcta entonces por la proposición 3.8 podemos concluir que M es un modelo para $STEMod(\mathbb{S}) \cup \{x = x\}$.

$\Leftarrow$) Como $STEMod(\mathbb{S}) \cup \{x = x\}$ tiene un modelo, entonces la proposición 3.7 garantiza que $STEMod(\mathbb{S}) \cup \{x = x, A\}$ tiene un E-modelo, digamos M, pero cada conjunto obtenido a partir de $STEMod(\mathbb{S})$ resolviendo con $x = x$ tiene también a M como modelo. De manera que en particular, M es un E-modelo de $\mathbb{S}$, por la proposición 3.9. $\dashv$

3.2.3 El Teorema de Completud

En esta sección demostraremos el teorema de completud para la paramodulación con RFP-resolución sin los axiomas funcionales reflexivos, si bien nos seguiremos basando en [Bra75], la incorporación de la RFP-resolución es una aportación nuestra que proporciona mayor claridad que el tratamiento original. La idea es transformar una refutación obtenida mediante hiperresolución del conjunto $\mathbb{S} \cup \{x = x\}$ a una refutación que sustituye en ciertos casos una aplicación de RFP-resolución por una aplicación de paramodulación.

Lema 3.2 *Sea* $\mathbb{S}$ *un conjunto de cláusulas aplanadas. Si* E *es un RFP-resolvente de cláusulas de* $\mathbb{S}$ *entonces* E *es aplanada.*

dem:

Sean $C \equiv C' \vee \neg Q$ y $D' \equiv D_1 \vee P \vee D_2$ tales que Q, P son átomos unificables mediante el umg σ y D' es variante de un factor de la cláusula positiva D y $C, D \in \mathbb{S}$. Queremos demostrar que $E \equiv (D_1 \vee D_2 \vee C')\sigma$ está aplanada. Puesto que D es positiva y aplanada, su factor D' no puede producir socios así que D' permanece aplanada. Analicemos ahora al RFP-resolvente E. Cada socio en E se obtiene aplicando σ a un socio de C (D no tiene socios pues es positiva); un socio en C es por hipótesis de la forma $s \neq w$ y para que σ mueva a w ésta tendría que figurar en una de las literales sobre las que se resolvió pero esto no puede ser pues éstas no tienen socios, ya que la resolución solo se aplica a la cláusula más a la derecha en C. Por lo tanto E es aplanada. $\dashv$

La siguiente definición nos proporciona una correspondencia unívoca que asigna a cada literal de una cláusula aplanada cierta expresión mediante la cual podremos transfomar una derivación dada.

Definición 3.22 Sean $\tilde{C}$ una cláusula aplanada y $\tilde{L}$ una literal de $\tilde{C}$, definimos recursivamente $\psi(\tilde{L})$ y L como sigue:
Si $\tilde{L}$ no tiene socios entonces $\tilde{L} \equiv s \neq w$ y definimos $\psi(\tilde{L}) \equiv s$. En otro caso, sean $\tilde{L_1}, \ldots, \tilde{L_n}$ todos los socios que figuran en $\tilde{L}$. Entonces, $\tilde{L_i} \equiv s_i \neq w_i$ y $\tilde{L} \equiv \tilde{L}[w_1, \ldots, w_n]$. Supongamos definidos L_i a partir de $\tilde{L_i}$. Entonces definimos L a partir de $\tilde{L}$ como $L \equiv \tilde{L}[\psi(L_1), \ldots, \psi(L_n)]$. Finalmente, si $\tilde{L}$ tiene nivel cero entonces definimos $\psi(\tilde{L}) \equiv L$ y si $\tilde{L}$ tiene nivel mayor que cero entonces L es de la forma $s \neq w$ y definimos $\psi(\tilde{L}) \equiv s$.
Por último extendemos ψ a cláusulas aplanadas como:

$$\psi(\tilde{C}) \equiv \bigvee\{\psi(\tilde{L}) \mid \tilde{L} \text{ figura en } \tilde{C} \text{ y } \tilde{L} \text{ tiene nivel cero}\}$$

Esta complicada definición se verá mejor con un ejemplo.

Ejemplo 3.9 Sea $\tilde{C} \equiv a \neq u \vee gx \neq v \vee f(u,v) \neq w \vee Pw \vee a \neq z \vee f(x,y) = gz$. Obtenemos $\psi(\tilde{L})$ para todas las literales de $\tilde{C}$:

1. $\psi(a \neq u) \equiv a$ pues $a \neq u$ no tiene socios.
2. $\psi(gx \neq v) \equiv gx$ pues $gx \neq v$ no tiene socios.
3. $\tilde{L} \equiv f(u,v) \neq w$ tiene como socios a $a \neq u, gx \neq v$ y tiene nivel mayor que cero, pues es socio de Pw. Así que $L \equiv (f(u,v) \neq w)[\psi(a \neq u), \psi(gx \neq v)] \equiv (f(u,v) \neq w)[a, gx] \equiv f(a,gx) \neq w$. $\therefore$ $\psi(f(u,v) \neq w) \equiv f(a,gx)$.
4. $\tilde{L} \equiv \psi(Pw)$ tiene nivel cero, así que $\psi(\tilde{L}) = L$, donde $L \equiv Pw[\psi(f(u,v) \neq w)] \equiv Pw[f(a,gx)] \equiv Pf(a,gx)$. $\therefore$ $\psi(Pw) \equiv Pf(a,gx)$.
5. $\psi(a \neq z) \equiv a$ pues $a \neq z$ no tiene socios.
6. $\tilde{L} \equiv f(x,y) = gz$ tiene nivel cero. En este caso $L \equiv (f(x,y) = gz)[\psi(a \neq z)] \equiv (f(x,y) = gz)[a] \equiv f(x,y) = ga$. $\therefore$ $\psi(f(x,y) = gz) \equiv f(x,y) = ga$.

La idea para la prueba de completud es reemplazar cada aplicación de RFP-resolución donde se resolvió un socio de una cláusula aplanada por una aplicación de paramodulación cuyo paramodulante es el RFP-resolvente.

Teorema 3.3 *(Completud de la Paramodulación).*
Sea $\mathbb{S}$ un conjunto de cláusulas que no tiene E-modelo. Entonces existe una refutación de $\mathbb{S} \cup \{x = x\}$ que sólo utiliza las reglas de RFP-resolución y Paramodulación.

dem:

Sea $\tilde{\mathbb{S}} \equiv EMod(\mathbb{S})$. Por el teorema 3.1 el conjunto $\tilde{\mathbb{S}} \cup \{A, x = x\}$ es no satisfacible. Así que, por la proposición 3.7 el conjunto $\tilde{\mathbb{S}} \cup \{x = x\}$ es no satisfacible, por lo tanto el teorema 2.7 garantiza que $\tilde{\mathbb{S}} \cup \{x = x\}$ tiene una refutación mediante

hiperresolución[5]. Dicha refutación se puede ver como una refutación utilizando RFP-resolución, pues así está definida la hiperresolución. Sea $\tilde{\Delta}$ una refutación de $\tilde{\mathbb{S}} \cup \{x = x\}$ que utiliza únicamente RFP-resolución. Puesto que $\tilde{\mathbb{S}} \cup \{x = x\}$ consiste de cláusulas aplanadas, el lema 3.2 garantiza que $\tilde{\Delta}$ consta de cláusulas aplanadas.
A partir de $\tilde{\Delta}$ vamos a definir una refutación Δ que utiliza paramodulación. Sea $\tilde{\Delta} \equiv (\tilde{C_1}, \ldots, \tilde{C_m})$ entonces definimos $\Delta \equiv (\psi(\tilde{C_1}), \ldots, \psi(\tilde{C_m}))$. Ahora falta definir las justificaciones de cada paso en la derivación Δ.
Con ayuda de la proposición 3.9 podemos ver fácilmente que si $\tilde{C} \in EMod(\mathbb{S})$ entonces $\psi(\tilde{C}) \equiv C$. Así que si $\tilde{C_i}$ se justificaba como elemento de $\tilde{\mathbb{S}}$ entonces $\psi(\tilde{C_i})$ se justifica como elemento de $\mathbb{S}$ en Δ.
Si $\tilde{C_i}$ se obtuvo mediante RFP-resolución de $\tilde{C_j}, \tilde{C_k}$ entonces hay dos casos:

1. Se resolvió un socio. Este caso corresponde a la paramodulación. La situación es como sigue: $\tilde{C_j} \equiv \tilde{C_j}' \vee \tilde{L}[w] \vee s' \neq w$, $\tilde{C_k} \equiv s = t \vee \tilde{C_k}'$ y entonces $\tilde{C_i} \equiv (\tilde{C_j}' \vee \tilde{L}[w] \vee \tilde{C_k})\sigma$,donde σ es umg de $\{s' = w, s = t\}$. Es decir, se aplicó RFP-resolución de la siguiente manera:

$$\frac{\tilde{C_j}' \vee \tilde{L}[w] \vee s' \neq w \qquad s = t \vee \tilde{C_k}'}{(\tilde{C_j}' \vee \tilde{L}[w] \vee \tilde{C_k})\sigma}$$

   Las imagenes respectivas de $\tilde{C_j}, \tilde{C_k}, \tilde{C_i}$ son $C_j \equiv C_j' \vee L[s'], C_k \equiv s = t \vee C_k'$, $C_i \equiv (C_j' \vee L[w] \vee C_k')\sigma$ de manera que el RFP-resolvente $\tilde{C_i}$ se mapea en un paramodulante desde C_k hacia C_j. Como $s'\sigma = s\sigma$, $w\sigma = t\sigma$ la paramodulación queda como sigue:

$$\frac{\begin{array}{l} s = t \vee C_k' \\ C_j' \vee L[s'] \end{array}}{(C_j' \vee L[t] \vee C_k')\sigma}$$

   Supongamos que este caso se ha realizado exhaustivamente y pasemos al siguiente caso.

2. Se resolvió una literal $\tilde{L}$ de nivel cero. Como ya no hay socios tenemos $\psi(\tilde{L}) \equiv \tilde{L}$ y todo queda igual en Δ.

Finalmente, como $\tilde{\Delta}$ es una refutación de $\tilde{\mathbb{S}} \cup \{x = x\}$ mediante RFP-resolución y claramente $\psi(\square) \equiv \square$, entonces Δ es una refutación de $\mathbb{S}$ mediante RFP-resolución y Paramodulación. ⊣

De manera que con el método de modificación hemos logrado probar la completud de la paramodulación junto con resolución, o en nuestro caso RFP-resolución. Una de las ventajas del método con respecto a la paramodulación es su completud junto con la estrategia del conjunto de soporte, cuestión que no debe ser difícil de mostrar con lo desarrollado hasta ahora: para mostrar

[5]Pues todo operador de refinamiento define un refinamiento, según el teorema 2.6.

la completud de paramodulación con conjunto de soporte necesitamos los axiomas reflexivos funcionales, una prueba de esto se encuentra en [WR70]. Una de las desventajas del método con respecto a la paramodulación es que resulta deductivamente más débil, como se muestra en el siguiente ejemplo

Ejemplo 3.10 A partir del conjunto $\mathbb{S} = \{a = b, Pfa, \neg Px \vee Qxx, x = x\}$ se puede derivar mediante paramodulación $Qfafb$. Mientras que a partir del conjunto modificado $\tilde{\mathbb{S}} = \{b \neq w \vee a = w, a \neq w \vee Pfw, \neg Px \vee Qxx, x = x\}$ resolución puede derivar $Qfafa, Qfbfb$ pero no $Qfafb$.

Actualmente el estudio de la paramodulación sigue en progreso. Para un panorama del estado actual de las investigaciones en este campo véase [McC97].

3.3 Reescritura de Términos

La habilidad para razonar con ecuaciones es importante en diversas aplicaciones en ciencias de la computación, incluyendo especificaciones algebraicas, lenguajes de programación de alto nivel y, por supuesto, razonamiento automático. El razonamiento sobre ecuaciones tiene que ver, por ejemplo, con el problema de decidir si una determinada ecuación se sigue como consecuencia lógica de un conjunto dado de ecuaciones, si una ecuación es verdadera en un modelo dado, o qué valores de las variables satisfacen una ecuación dada.

3.3.1 Sistemas Ecuacionales

Definición 3.23 Una *ecuación* es una fórmula atómica del tipo $s = t$, donde s, t son términos de la signatura dada.

Definición 3.24 Un *sistema ecuacional* es un conjunto de ecuaciones.

Frecuentemente estamos interesados en saber si una ecuación es consecuencia lógica de un sistema ecuacional dado. Por ejemplo, tal vez quisieramos saber si todo grupo de exponente 2 es abeliano, es decir, para el no iniciado en teoría de grupos, si la ecuación $x * y = y * x$ es consecuencia lógica del sistema ecuacional $\{x * e = x, x * i(x) = e, (x * y) * z = x * (y * z), x^2 = e\}$. Los sistemas ecuacionales son de interés para las ciencias de la computación así como para las matemáticas. Estructuras de datos muy comunes como listas y stacks (pilas) se pueden describir mediante un sistema ecuacional.

Ejemplo 3.11 El siguiente sistema ecuacional describe el tipo de datos pila (stack):

pop(push(y,x))=y.

pop(nil)=nil.

top(nil)=0.

top(push(y,x))=x.

Los sistemas para mecanizar el razonamiento ecuacional en una computadora se están volviendo cada dia más poderosos. Por otra parte un programa funcional es esencialmente un sistema ecuacional, que incluye típicamente funciones de orden superior, y la ejecución del programa es entonces un tipo de razonamiento ecuacional. Para una introducción avanzada al tema sugerimos consultar [Klo92, Pla94].

Definición 3.25 Sea E un sistema ecuacional. Definimos la relación $\rightarrow_E$ entre términos, como sigue:

$$s \rightarrow_E t \text{ syss existe } l = r \in E \text{ tal que } s = w[l\sigma] \text{ y } t = w[r\sigma]$$

para algun término w y sustitución σ. Donde la notación $w[t]$ significa que el término w contiene a t como subtérmino.
A partir de $\rightarrow_E$ definimos las siguientes relaciones:

$\rightarrow_E^+$ es la cerradura transitiva de $\rightarrow_E$.

$\rightarrow_E^*$ es la cerradura reflexiva y transitiva de $\rightarrow_E$.

$\leftarrow_E$ es la inversa de $\rightarrow_E$.

$\leftrightarrow_E$ es $\rightarrow_E \cup \leftarrow_E$.

Para una definición más detallada de los conceptos anteriores véase el apéndice de [SA91].

La relación $\rightarrow_E$ tiene relación con el llamado *Cálculo de Birkhoff*.
El Cálculo de Birkhoff B es un subcálculo del cálculo de Hilbert con igualdad, que consta del esquema de axioma de reflexividad $t = t$ y de las siguientes reglas de inferencia:

$$\frac{s = t}{t = s} \text{ Simetría} \qquad \frac{s = t \;\; t = r}{s = r} \text{ Transitividad}$$

$$\frac{s_1 = t_1, \ldots, s_n = t_n}{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)} \text{ Congruencia} \qquad \frac{s = t}{s\sigma = t\sigma} \text{ Sustitución}$$

Ejemplo 3.12 Sea E el sistema ecuacional del ejemplo 3.11.Entonces $E \vdash_B push(s,1) = push(pop(push(s,0)),1)$ mediante la siguiente derivación:

$pop(push(s,x)) = s$	Elemento de E
$pop(push(s,0)) = s$	Sustitución
$1 = 1$	Axioma
$push(pop(push(s,0)),1) = push(s,1)$	Congruencia
$push(s,1) = push(pop(push(s,0)),1)$	Simetría

B es un cálculo completo y correcto.

Teorema 3.4 *(Birkhoff, 1935)*
Sea E un sistema ecuacional. Entonces para cualesquiera términos s, t:

$$E \models s = t \text{ syss } E \vdash_B s = t$$

dem:

Véase [SA91] ⊣

Este teorema proporciona un método para derivar las consecuencias lógicas de un sistema ecuacional. Sin embargo es ineficiente, desde el punto de vista computacional, por eso estamos interesados en encontrar restricciones de las reglas de B que sean capaces de derivar las consecuencias lógicas de un sistema ecuacional dado. Esto es la motivación para la teoría de sistemas de reescritura de términos.
La ventaja de la relación $\rightarrow_E$ resalta en este momento al compararla con la derivación que acabamos de hacer, pues $push(pop(push(s,0)),1) \rightarrow_E push(s,1)$ lo cual es claramente más simple de obtener que la derivación. En la siguiente sección quedara más clara la importancia de la relación $\rightarrow_E$.

3.3.2 Sistemas de Reescritura de Términos

La idea de un sistema de reescritura de términos (s.r.t.) es orientar una ecuación $r = s$ convirtiéndola en una regla $r \rightarrow s$, la cual indica que las instancias del término r pueden reemplazarse por instancias de s pero no al revés.
De qué manera orientar reglas y qué condiciones garantizan que los sistemas resultantes tengan el mismo poder computacional que los sistemas ecuacionales de los que provienen, son dos de los principales tópicos de estudio en el área de reescritura de términos.

Definición 3.26 Un *Sistema de Reescritura de Términos* R sobre un conjunto de términos $\mathcal{T}$ es un conjunto de ecuaciones orientadas, llamadas reglas de reescritura. Cada una de ellas es de la forma $l \rightarrow r$ donde $l, r \in \mathcal{T}$. De manera que un s.r.t. se puede ver como una relación binaria sobre $\mathcal{T}$.

De manera análoga a la relación $\rightarrow_E$ se definen las relaciones $\rightarrow_R$, $\rightarrow_R^+$, $\rightarrow_R^*$ $\leftarrow_R$, $\leftrightarrow_R$ para cualquier s.r.t. R.

Una primera relación entre un s.r.t. y un sistema ecuacional es la siguiente. Dado un s.r.t. R definimos el *sistema ecuacional asociado a R*, E_R como $E_R = \{s = r \mid s \rightarrow r \in R\}$, el corolario al siguiente teorema relaciona a R con E_R.

Teorema 3.5 *Sean $s = t$ una ecuación, R un s.r.t y E_R el sistema ecuacional asociado a R. Entonces*

$$E_R \models s = t \text{ syss } s \leftrightarrow_{E_R}^* t$$

dem:

Ver [SA91] ⊣

Corolario 3.1 $E_R \models s = t$ *syss* $s \leftrightarrow_R^* t$

dem:

Es trivial. ⊣

Aún cuando ya tenemos una relación entre los dos conceptos, el hecho de que la reescritura pueda ir en ambas direcciones en $s \leftrightarrow_R^* t$ es una desventaja puesto que la completud del sistema no genera un procedimiento de decisión óptimo. Lo que mostraremos es que si R tiene ciertas propiedades entonces $E_R \models s = t$ syss cualquier forma normal de s es idéntica a cualquier forma normal de t. Esto genera un procedimiento que permite decidir la consecuencia lógica reescribiendo ambos términos en formas normales y checando si éstas son idénticas sintácticamente.

Las siguientes definiciones son para cualquier relación binaria $\rightarrow \subseteq \mathcal{T} \times \mathcal{T}$.

Definición 3.27 Un termino t es *irreducible* si no puede ser reescrito. Si $r \rightarrow^* s$ y s es irreducible decimos que s es una *forma normal* de r o que r se *reduce* a s.

Definición 3.28 La relación binaria $\rightarrow$ tiene la *propiedad de Church-Rosser* si y sólo si $\leftrightarrow^* \subseteq \rightarrow^* \circ \leftarrow^*$. Es decir, si $s \leftrightarrow^* t$ implica que existe $r \in \mathcal{T}$ tal que $s \rightarrow^* r \leftarrow^* t$.

Definición 3.29 La relación binaria $\rightarrow$ es *confluente* si y sólo si $\leftarrow^* \circ \rightarrow^* \subseteq \rightarrow^* \circ \leftarrow^*$. Es decir, para cualesquiera términos r, s, t, si $s \leftarrow^* r \rightarrow^* t$ entonces existe un término l tal que $s \rightarrow^* l \leftarrow^* t$.

Diremos que un s.r.t. R es Church-Rosser (Confluente) en el caso en que $\rightarrow_R$ sea Church-Rosser (Confluente). Uno de los resultados básicos de la teoría de reescritura de términos es la equivalencia entre las dos propiedades anteriores; para facilitar la demostración de tal hecho introducimos las siguientes relaciones auxiliares:

$$s \downarrow t \quad \text{syss} \quad s \rightarrow^* r \leftarrow^* t \quad \text{para algún } r \in \mathcal{T}$$

$$s \uparrow t \quad \text{syss} \quad s \leftarrow^* r \rightarrow^* t \quad \text{para algún } r \in \mathcal{T}$$

Con estas relaciones auxiliares, las propiedades de Church-Rosser y confluencia quedan enunciadas como sigue:

$\rightarrow$ es Church-Rosser syss $s \leftrightarrow^* t$ implica $s \downarrow t$.

$\rightarrow$ es Confluente syss $s \uparrow t$ implica $s \downarrow t$.

Los siguientes lemas son necesarios para la demostración del teorema 3.6; con $\downarrow^n$ denotamos la composición de $\downarrow$ consigo misma n veces.

Lema 3.3 *Si R es confluente y $s \downarrow^2 t$ entonces $s \downarrow t$.*

dem:

Como $s \downarrow^2 t$ entonces existe un término r tal que $s \downarrow r$ y $r \downarrow t$. Así que existen términos q_1, q_2 tales que $s \rightarrow^* q_1 \leftarrow^* r$ y $r \rightarrow^* q_2 \leftarrow^* t$, lo cual implica que $q_1 \leftarrow^* r \rightarrow^* q_2$, es decir, $q_1 \uparrow q_2$. Ahora bien, ya que R es confluente y $q_1 \uparrow q_2$ entonces $q_1 \downarrow q_2$. Por lo tanto existe un término w tal que $q_1 \rightarrow^* w \leftarrow^* q_2$, de donde obtenemos $s \rightarrow^* q_1 \rightarrow^* w \leftarrow^* q_2 \leftarrow^* t$ y en conclusión $s \downarrow t$. ⊣

Lema 3.4 *Si $s \leftrightarrow^* t$ entonces $s \downarrow^n t$ para alguna $n \in \omega$.*

dem:

Obsérvese que para cualesquiera términos r, u, si $r \rightarrow^* u$ ó $u \rightarrow^* r$ entonces $r \downarrow u$ debido a que $r \rightarrow^* r$ y $u \rightarrow^* u$, puesto que entonces se obtiene $r \rightarrow^* u \leftarrow^* u$ o $r \rightarrow^* r \leftarrow^* u$, es decir, $r \downarrow u$. Así que por lo anterior y por la definición de $\leftrightarrow^*$, dado que existen términos $r_1, \ldots, r_m$ tales que $s \leftrightarrow^* r_1 \leftrightarrow^* \ldots \leftrightarrow^* r_m \leftrightarrow^* t$, podemos concluir que existe una $n \in \omega$ tal que $s \downarrow^n t$. ⊣

Con la ayuda de los dos lemas anteriores podemos probar la equivalencia anunciada.

Teorema 3.6 *Si R es un s.r.t. entonces R es Church-Rosser si y sólo si R es confluente.*

dem:

Sea $\rightarrow^* = \rightarrow_R^*$.
$\Rightarrow$) Tomemos dos términos tales que $s \uparrow t$, lo que queremos ver es que $s \downarrow t$. Por hipótesis existe un término r tal que $s \leftarrow^* r \rightarrow^* t$.
$r \rightarrow^* s$ implica $r \leftrightarrow^* s$ y $r \rightarrow^* t$ implica $r \leftrightarrow^* t$; finalmente $s \leftrightarrow^* r \leftrightarrow^* t$ por lo que $s \leftrightarrow^* t$ y como R es Church-Rosser entonces $s \downarrow t$.
$\Leftarrow$) Tomemos dos términos tales que $s \leftrightarrow^* t$. Queremos demostrar que $s \downarrow t$. Por el lema 3.4 tenemos que $s \downarrow^n t$ para alguna $n \in \omega$. El caso no trivial es cuando $n \geq 2$.
Mediante inducción sobre n mostraremos que si $s \downarrow^n t$ y $n \geq 2$ entonces $s \downarrow t$. La base de la inducción resulta inmediata a partir del lema 3.3. Supóngase que $s \downarrow^{n+1} t$ y $n \geq 1$, en tal caso existe un término r tal que $s \downarrow^2 r \downarrow^{n-1} t$. Usando el lema 3.3 tenemos que $s \downarrow r$, por lo que $s \downarrow^n t$ y por HI podemos concluir que $s \downarrow t$. ⊣

Definición 3.30 Un s.r.t R es *terminal* si no tiene sucesiones infinitas de reescritura, es decir si la relación $\rightarrow_R$ es bien fundada.

Informalmente lo anterior significa que el proceso de reescritura aplicado a un término terminará eventualmente sin importar cómo se aplicaron las reglas de reescritura. Los sistemas terminales también se conocen como sistemas *fuertemente normalizables* o *Noetherianos*.

Ejemplo 3.13 El sistema $R = \{g(x) \rightarrow f(x), f(x) \rightarrow x\}$ es terminal.
Esto es claro, pues la primera regla, que cambia las presencias de g por f, sólo puede ser aplicada tantas veces como presencias de g haya en el término y la segunda regla sólo se puede aplicar tantas veces como presencias de f tenga el término.
Como ejemplo de sistema no terminal tenemos a $R = \{x \rightarrow f(x)\}$

Definición 3.31 Un *orden de terminación* para los términos es un orden parcial $>$ que satisface las siguientes propiedades:

1. $>$ es bien fundado.
2. Propiedad de Invariancia Total: Sea σ una sustitución. Si $s > t$ entonces $s\sigma > t\sigma$.
3. Propiedad de Reemplazo: Si $s > t$ entonces $f(\ldots s \ldots) > f(\ldots t \ldots)$

Los órdenes de terminación son una herramienta para mostrar si un sistema es terminal, como lo asegura el siguiente teorema.

Teorema 3.7 *Sean R un s.r.t y $>$ un orden de terminación. Si $R \subseteq >$ entonces R es terminal.*

dem:

Esto es inmediato puesto que $>$ es bien fundado, de manera que la existencia de una R-sucesión infinita decreciente implicaria la existencia de una $>$-sucesión infinita decreciente, lo cual es absurdo. ⊣

Si R es terminal siempre podemos encontrar una forma normal de cualquier término, pues toda secuencia de reescritura termina de manera que el último elemento será una forma normal. Sin embargo, un término puede tener más de una forma normal.
Si R es terminal y además confluente, cualquier término tendrá siempre una y sólo una forma normal.

Definición 3.32 Si R es terminal y confluente decimos que R es *canónico*.

Aún no tenemos herramienta suficiente para mostrar si un sistema es confluente, por lo pronto damos un ejemplo de s.r.t no confluente.

Ejemplo 3.14 Considerese el s.r.t. R que consta de las siguientes reglas:

(1) $not(not(x)) \rightarrow x$

(2) $not(or(x,y)) \rightarrow and(not(x), not(y))$

(3) $not(and(x,y)) \rightarrow or(not(x), not(y))$

(4) $and(x, or(y,z)) \rightarrow or(and(x,y), and(x,z))$

(5) $and(or(x,y), z) \rightarrow or(and(x,z), and(y,z))$

este sistema convierte una expresión booleana a forma normal disyuntiva.
Es claro que R es terminal pues las negaciones y conjunciones se van recorriendo a la derecha, proceso que sólo puede realizarse un número finito de veces. Sin embargo R no es canónico ya que no es confluente. El término $and(or(x,y), or(u,v))$ tiene dos formas normales distintas:

$$or(or(and(x,u), and(x,v)), or(and(y,u), and(y,v)))$$

$$or(or(and(x,u), and(y,u)), or(and(x,v), and(y,v)))$$

aunque las dos formas normales son equivalentes bajo asociatividad y conmutatividad del *and*. Tal vez quisieramos agregar las reglas de asociatividad y conmutatividad del *and*, pero el agregar reglas para la conmutatividad causaría la pérdida de terminalidad.

Si tenemos un sistema canónico R entonces ya tenemos un proceso de decisión para la teoría ecuacional subyacente E_R. Para ello, basta seguir los siguientes pasos para contestar la pregunta ¿ $E_R \models s = t$?:

1. Reducir s y t a sus respectivas formas normales s' y t'
2. Comparar s' y t'. $E_R \models s = t$ syss $s' = t'$.

Aun no hemos dicho como probar la propiedad de confluencia. En la forma como se ha enunciado resulta una propiedad complicada pero si R es terminal la confluencia es decidible mediante la confluencia local, como lo asegura el lema de Newman que demostramos a continuación.

Definición 3.33 Una relación binaria $\rightarrow$ es *localmente confluente* si para cualesquiera términos r, s, t, si $s \leftarrow r \rightarrow t$ entonces $s \downarrow t$.

Resulta obvio que si R es confluente entonces R es localmente confluente. Si además R es terminal entonces tenemos la equivalencia.

Teorema 3.8 *(Lema de Newman)*
Sea R un s.r.t terminal. Si R es localmente confluente entonces R es confluente.

dem:

Como R es terminal entonces $\rightarrow_R$ es bien fundada. Por lo que podemos aplicar $\rightarrow$-inducción para probar que si $s \uparrow t$ entonces $s \downarrow t$.
Hipótesis de Inducción: Si $u \rightarrow u'$ y $s \leftarrow^* u' \rightarrow^* t$ entonces $s \downarrow t$.

Paso Inductivo: Supongamos que $s \leftarrow^* u \rightarrow^* t$. El caso no trivial es cuando u es distinto de s y t. En tal caso, por definición de $\rightarrow^*$ existen términos s_1, t_1 tales que $s \leftarrow^* s_1 \leftarrow u \rightarrow t_1 \rightarrow^* t$. Como R es localmente confluente existe un término v tal que $s_1 \rightarrow^* v \leftarrow^* t_1$. Aplicando la HI a s_1 (puesto que $u \rightarrow s_1$ y $s \leftarrow^* s_1 \rightarrow^* v$) obtenemos un término w tal que $s \rightarrow^* w \leftarrow^* v$. De la misma manera, usando la HI con t_1 (ya que $u \rightarrow t_1$ y $t \leftarrow^* t_1 \rightarrow^* w$), obtenemos un término r tal que $t \rightarrow^* r \leftarrow^* w$. Finalmente tenemos $s \rightarrow^* w \rightarrow^* r \leftarrow^* t$, es decir, $s \downarrow t$. ⊣

3.3.3 Pares Criticos

Hasta este momento hemos reducido el problema de la confluencia al de la confluencia local para un sistema terminal, pero todavia nos queda por establecer cuando un sistema es localmente confluente. Esto lo haremos utilizando el concepto de pares criticos. Para mostrar que un s.r.t es localmente confluente necesitamos saber que si un término se reescribe de dos maneras distintas, entonces éstas tienen una forma normal común.
Primero vamos a considerar las relaciones entre las diferentes reescrituras de un término.

Definición 3.34 Sean R un s.r.t., t un término y u un subtérmino de t. Si u es instancia del lado izquierdo de una regla de R entonces u es un *redex* de t.

Cada forma de reescribir un término t corresponde a un redex de t. Nos interesa analizar que sucede si un término se reescribe de dos maneras distintas. Existen 3 casos que ejemplificamos enseguida:

1. Los redexes para la reescritura son subtérminos ajenos de t.

t es $f(a,b)$	$R = \{a \rightarrow c, b \rightarrow d\}$
Los redexes son	a, b
Primer paso de reescritura:	$f(c,b)$ ó $f(a,d)$
Segundo paso de reescritura:	$f(c,d)$ ó $f(c,d)$

2. Un redex es subtérmino del otro.

t es $g(a)$	$R = \{a \to b, g(x) \to f(x,x)\}$
Los redexes son	$a, g(a)$
Primer paso de reescritura:	$g(b)$ ó $f(a,a)$
Segundo paso de reescritura:	$f(b,b)$ ó $f(b,b)$

en este caso el redex a es subtérmino del redex $g(a)$ y corresponde a la posición de la variable x en la regla $g(x) \to f(x,x)$. En general, esta posibilidad sucede cuando un redex es subtérmino del otro y corresponde a la posición de una variable en otra regla o bien cuando es subtérmino del tal posición.

3. Un redex es subtérmino del otro pero no corresponde a la posición de una variable en otra regla.

t es $(x * e) * z$	$R = \{(x * y) * z \to x * (y * z), x * e \to x\}$
Los redexes son	$(x * e) * z, x * e$
Primer paso de reescritura:	$x * (e * z)$ ó $x * z$
Segundo paso de reescritura:	No es posible.

Estas son las únicas posibilidades para que un término se reescriba de dos maneras diferentes. Las dos primeras siempre producen un par de términos con una forma normal común. El tercer caso es el que viola la confluencia local. En este caso se dice que las *reglas se traslapan* o que ha ocurrido un *traslapamiento* y el par de términos resultantes es un *par crítico*. A continuación formalizamos todo esto.

Definición 3.35 Sea t un término, definimos la *cadena de* t, denotada $\overline{\overline{t}}$, como la sucesión de símbolos de t sin paréntesis ni comas. Si α es $\overline{\overline{t}}$ para algún t entonces t es único y definimos $\underline{\underline{\alpha}}$ como t.

Ejemplo 3.15 Si t es $f(x, g(y))$ entonces $\overline{\overline{t}}$ es $fxgy$ e inversamente $\underline{\underline{fxgy}}$ es $f(x, g(y))$. Obsérvese que no hay ambigüedad al obtener $\underline{\underline{fxgy}}$ pues la aridad de los símbolos está determinada de manera única por la signatura.

Definición 3.36 Sea s un término y t una presencia de algún subtérmino de s. La descomposición de $\overline{\overline{s}}$ en $\alpha\overline{\overline{t}}\gamma$ se denota con $(\alpha, \overline{\overline{t}}, \gamma)$. El par (α, γ) es el *contexto* de t.

Es fácil ver que si α, β son presencias de subtérminos de $\overline{\overline{t}}$ entonces son ajenas o una es subtérmino de la otra.

Definición 3.37 Si $\overline{\overline{t}}$ es $\gamma_1 x_1 \gamma_2 x_2 \ldots \gamma_n x_n \gamma_{n+1}$ donde las γ_i no tienen variables y σ es una sustitución tal que $\overline{\overline{t\sigma}}$ es $\gamma_1 \overline{\overline{t_1}} \gamma_2 \overline{\overline{t_2}} \ldots \gamma_n \overline{\overline{t_n}} \gamma_{n+1}$ decimos que la presencia del subtérmino t_i es una *presencia de variable instanciada.*

Ahora analizamos formalmente las tres posibilidades anteriores. Supongamos que t tiene dos redexes $l_1\sigma$ y $l_2\sigma$ y digamos que $\overline{\overline{l_1}}$ es $\gamma_1 x_1 \gamma_2 x_2 \ldots \gamma_n x_n \gamma_{n+1}$. Supongamos que $l_1\sigma$ es el redex de mayor longitud (si ambos son de la misma longitud no importa) y que $\overline{\overline{l_1\sigma}}$ es $\gamma_1 \overline{\overline{t_1}} \gamma_2 \overline{\overline{t_2}} \ldots \gamma_n \overline{\overline{t_n}} \gamma_{n+1}$. Entonces $\overline{\overline{t}}$ resulta ser $\alpha\gamma_1 \overline{\overline{t_1}} \gamma_2 \overline{\overline{t_2}} \ldots \gamma_n \overline{\overline{t_n}} \gamma_{n+1}\beta$ para algunas cadenas α, β.

El redex $l_2\sigma$ puede figurar en 3 lugares:

1. Puede ser subcadena de α o de β, lo cual significa que los redexes son ajenos.
2. Puede ser subcadena de algún t_i
3. Puede ser subcadena del redex $l_1\sigma$ y no subcadena de t_i para toda i. Este es el único caso que corresponde a un par crítico.

No hay más posibilidades pues, como ya se dijo, dos presencias de subtérminos son ajenas o una es subtérmino de la otra. Por fin podemos dar la definición formal de par crítico.

Definición 3.38 Sean R un s.r.t., $l_1 \to r_1, l_2 \to r_2$ dos reglas de R sin variables en común, u un subtérmino de l_1 que no es una variable. Supóngase que u y l_2 son unificables mediante el umg σ y que $\overline{\overline{l_1}}$ es $\gamma_1 u \gamma_2$. El par $(\underline{\overline{\gamma_1 r_2 \gamma_2}}\sigma, r_1\sigma)$ es un *par crítico* para $l_1 \to r_1, l_2 \to r_2$ o para R. u es un *subtérmino con presencia crítica* de l_1 y σ es la *sustitución crítica.*

En esta definición tenemos las siguientes observaciones: u podría ser l_1 (a menos que l_1 sea una variable), las reglas $l_1 \to r_1, l_2 \to r_2$ son distintas o u no es l_1 (esto prohibe pares críticos triviales de una regla consigo misma).

Ejemplo 3.16 Considérese el sistema $R = \{g(u)*u \to 1, (x*y)*z \to x*(y*z)\}$. El término $g(u) * u$ se puede unificar con el subtérmino $x * y$ de $(x * y) * z$ obteniendose el término $(g(u) * u) * z)$. Este término se reescribe como $1 * z$ usando la primera regla y como $g(u) * (u * z)$ usando la segunda regla. Esto produce el par crítico $(g(u) * (u * z), 1 * z)$.

Ejemplo 3.17 Las reglas

$$f(f(u,v), g(v,u)) \to h(u,v),\ g(f(x,y), g(y,x)) \to k(x,y)$$

tienen cuatro pares críticos, obtenidos unificando los lados izquierdos con un subtérmino de ellos mismos o con un subtérmino del otro lado izquierdo. Uno de ellos es $(g(h(u,v), g(g(v,u), f(u,v))), k(f(u,v), g(v,u)))$ el cual se obtiene unificando el lado izquierdo de la primera regla con el subtérmino $f(x,y)$ del lado izquierdo de la segunda regla.

Con el siguiente teorema finalmente damos una manera para probar si un sistema R es confluente. Esta demostración es aportación nuestra basada en el bosquejo dado en [Pla94].

Teorema 3.9 *(Teorema de los pares críticos)*
Sea R un s.r.t terminal. R es confluente si y sólo si para todos los pares críticos (s,t) de R y para formas normales arbitrarias s', t' de s, t respectivamente, s' y t' son idénticas.

dem:

$\Rightarrow$) Supongamos que R es confluente. En tal caso como R también es terminal, todo término tiene exactamente una forma normal. Dado cualquier par crítico (s,t), se tiene que $s \uparrow t$, por lo que la confluencia nos lleva a $s \downarrow t$. Ahora bien, sea u tal que $s \to^* u \leftarrow^* t$, puesto que s' y t' son formas normales únicas entonces tenemos que $s \to^* u \to^* s'$ y $t \to^* u \to^* t'$, de donde se concluye que $s' \uparrow t'$ y utilizando la confluencia tenemos $s' \downarrow t'$ lo cual implica que s' y t' son idénticas.

$\Leftarrow$) Supongamos que se cumple tal propiedad. Queremos demostrar que R es confluente, es decir, si $s \uparrow t$ entonces $s \downarrow t$. Basta analizar que se cumple la propiedad para un solo paso de reescritura, es decir, basta probar la confluencia local pues tenemos el lema de Newman. Digamos que el término r se reescribe en r_1 y en r_2, $r_2 \leftarrow r \to r_1$ (es decir, $r_1 \uparrow r_2$). Vamos a mostrar entonces que $r_1 \downarrow r_2$. Tenemos que analizar los tres casos ya expuestos:
Sean $l_1 \to m_1, l_2 \to m_2 \in R$ y σ_1, σ_2 sustituciones ajenas tales que el redex para r_i es $l_i\sigma_i$.

1. Los redexes para $r \to r_1$ y $r \to r_2$ son ajenos.
   Este es el caso trivial. $\overline{r}$ es $\alpha\overline{l_1\sigma_1}\beta\overline{l_2\sigma_2}\gamma$ para algunas cadenas α, β, γ. Así que $\overline{r_1}$ es $\alpha\overline{m_1\sigma_1}\beta\overline{l_2\sigma_2}\gamma$ y $\overline{r_2}$ es $\alpha\overline{l_1\sigma_1}\beta\overline{m_2\sigma_2}\gamma$, de manera que si aplicamos la primera regla a r_2 y la segunda regla a r_1 obtenemos en ambos casos $\alpha\overline{m_1\sigma_1}\beta\overline{m_2\sigma_2}\gamma$. Por lo tanto $r_1 \downarrow r_2$.

2. El redex para $r \to r_2$ es subtérmino de una presencia de variable instanciada para $r \to r_1$ (o viceversa).
   Tenemos que $\overline{r}$ es $\alpha\overline{l_1\sigma_1}\beta$ para algún contexto (α, β) y $\overline{l_1\sigma_1}$ es $\gamma\overline{l_2\sigma_2}\delta$ para algún contexto (γ, δ). Analizando a $l_1\sigma_1$ tenemos que $\overline{l_1\sigma_1}$ se reescribe en $\overline{m_1\sigma_1}$ mediante la primera regla y en $\gamma\overline{m_2\sigma_2}\delta$ mediante la segunda regla. Por lo tanto basta ver que $m_1\sigma_1$ y $\gamma m_2\sigma_2\delta$ se pueden reescribir en un término común.
   Puesto que $l_2\sigma_2$ es presencia de variable instanciada entonces hay una variable x tal que $\overline{l_1}$ es $\gamma' x \delta'$ donde $\gamma'\sigma_1 = \gamma, \delta'\sigma_1 = \delta, x\sigma_1 = l_2\sigma_2$. Definimos la sustitución σ_3 como sigue: $x\sigma_3 = m_2\sigma_2$ y si $y \neq x$, $y\sigma_3 = y\sigma_1$.
   Claramente $x\sigma_1 = l_2\sigma_2 \to m_2\sigma_2 = x\sigma_3$ y de esto se sigue que $m_1\sigma_1 \to^* m_1\sigma_3$.
   Veamos ahora que $\gamma m_2\sigma_2\delta \to^* m_1\sigma_3$. $\gamma m_2\sigma_2\delta$ se obtuvo a partir de $l_1\sigma_1$

reemplazando la imagen de una presencia de x bajo σ_1 por $m_2\sigma_2$ pero podría haber más presencias de x en l_1. Si tomamos todas las presencias de x en l_1 y también las reemplazamos por $m_2\sigma_2$ obtenemos $l_1\sigma_3$. Por lo tanto $l_1\sigma_1 \to^* l_1\sigma_3 \to m_1\sigma_3$ de donde se concluye que $\gamma m_2\sigma_2\delta \to^* m_1\sigma_3$. De todo lo anterior podemos concluir que $r_1 \downarrow r_2$

3. El redex para $r \to r_2$ es subtérmino del redex para $r \to r_1$ (o viceversa) pero no es presencia de variable instanciada para $r \to r_1$. Este es el caso crítico.
   En esta situación existe un contexto (α, δ) y un subtérmino u de l_1 tal que $\overline{\overline{l_1}}$ es $\alpha\overline{\overline{u}}\delta$ y $\overline{\overline{l_1\sigma_1}}$ es $\alpha'\overline{\overline{u\sigma_1}}\delta'$ donde $\alpha\sigma_1 = \alpha'$, $\delta\sigma_1 = \delta'$ y $u\sigma_1 = l_2\sigma_2$. Dado que σ_1, σ_2 son ajenas, $\sigma' = \sigma_1 \cup \sigma_2$ esta bien definida y cumple que $u\sigma' = u\sigma_1 = l_2\sigma_2 = l_2\sigma'$. Por lo tanto u y l_2 son unificables. Sea σ un umg de $\{u, l_2\}$.
   El par $(\overline{\overline{\alpha m_2\delta\sigma}}, m_1\sigma)$ es un par crítico y como R es terminal, existen formas normales n_1, n_2 de $\overline{\overline{\alpha m_2\delta\sigma}}, m_1\sigma$ respectivamente, que por hipótesis son idénticas. Esto permite concluir, de manera análoga al caso anterior, que $r_1 \downarrow r_2$.

⊣

Corolario 3.2 *(Knuth-Bendix,1970). Sea R un s.r.t. terminal. R es confluente syss todo par crítico (s,t) de R converge, es decir, $s \downarrow t$.*

dem:

Es inmediato. ⊣

Este resultado implica que si R es un s.r.t. finito y terminal, entonces la confluencia de R es decidible pues sólo hay un número finito de pares críticos. Veamos un par de ejemplos.

Ejemplo 3.18 Considerese el sistema R que consiste de las siguientes tres reglas que modelan las propiedades que definen un monoide:

(1) $((x * y) * z) \to (x * (y * z))$

(2) $(x * e) \to x$

(3) $(e * y) \to y$

Es claro que R es terminal pues las reglas 2 y 3 reducen el número de presencias de e en un término y la regla 1 mueve paréntesis hacia la derecha, por lo tanto cada regla sólo puede aplicarse un número finito de veces. Para ver que R es Church-Rosser vamos a analizar todos los pares críticos de R. Para ello consideramos cada subtérmino que no es una variable del lado izquierdo de una regla que sea unificable con el lado izquierdo de una de las reglas.

1. Consideremos la ecuación (1). $((x * y) * z)$ tiene como subtérmino no variable a $(x * y)$ que puede unificarse con el lado izquierdo de cualquier regla:

   (a) Unificando $(x * y)$ con el lado izquierdo de (1), después de renombrar variables, digamos x, y, z por x_1, y_1, z_1, obtenemos el par crítico $(((x_1 * y_1) * (z_1 * z)), ((x_1 * (y_1 * z_1)) * z))$ mediante el umg $\sigma = \{x/(x_1 * y_1), y/z_1\}$. Pero tal par converge pues

   $$((x_1 * y_1) * (z_1 * z)) \rightarrow (x_1 * (y_1 * (z_1 * z)))$$

   $$((x_1 * (y_1 * z_1)) * z) \rightarrow (x_1 * ((y_1 * z_1) * z)) \rightarrow (x_1 * (y_1 * (z_1 * z)))$$

   (b) Unificando $(x * y)$ con el lado izquierdo de (2), después de renombrar x por x_1, mediante $\sigma = \{x/x_1, y/e\}$, producimos el par crítico $(x_1 * (e * z), x_1 * z))$ y tal par converge pues $x_1 * (e * z) \rightarrow_3 x_1 * z$.

   (c) Unificando $(x * y)$ con el lado izquierdo de (3), después de renombrar y por y_1 obtenemos el par crítico $((e * (y_1 * z)), (y_1 * z))$ mediante $\sigma = \{x/e, y/y_1\}$. Dicho par converge pues $(e * (y_1 * z)) \rightarrow (y_1 * z)$.

2. Consideremos la ecuación (2). Hay dos subtérminos no variables de $(x * e)$, que son e y $(x * e)$.

   (a) e no es unificable con ninguno de los lados izquierdos de las reglas, por lo que no produce pares críticos.

   (b) Si unificamos $(x * e)$ con $((x_1 * y_1) * z_1)$ obtenemos el par crítico $((x_1 * (e * z_1)), (x_1 * z_1))$ que obviamente converge.

   (c) Si unificamos $(x * e)$ con $(e * y)$ obtenemos el par crítico trivial (e, e).

3. La ecuación (3) se analiza de manera análoga a la (2).

Por lo tanto por el corolario 3.2 podemos garantizar que R es un sistema canónico.

Ejemplo 3.19 Para representar la función de concatenación en listas tenemos el sistema R que consta de

(1) $append(cons(x, y), z) \rightarrow cons(x, append(y, z))$

(2) $append(nil, z) \rightarrow z$

claramente R es terminal. Además es fácil ver que no tiene pares críticos. Por lo tanto R es canónico.

3.4 El Método de Knuth-Bendix

El método de compleción de Knuth-Bendix [KB70] fue concebido originalmente como un medio para transformar una axiomatización de alguna teoría ecuacional y generar a partir de ella un sistema de reescritura que se pudiera utilizar para decidir la validez de ecuaciones en la teoría dada. Sin embargo actualmente tiene diversas aplicaciones, como la generación de soluciones de ecuaciones, la síntesis de programas recursivos y, dentro del razonamiento automático, la demostración de teoremas acerca de estructuras definidas recursivamente y en general la demostración de teoremas del cálculo de predicados. En esta sección sólo nos limitaremos a describir el método; en lo que respecta a las aplicaciones recomendamos consultar [Der89].
Antes de dar el proceso formal, presentamos un ejemplo informal.
Consideremos el sistema ecuacional para la teoría de grupos $E = \{e \cdot x = x,\ i(x) \cdot x = e,\ (x \cdot y) \cdot z = x \cdot (y \cdot z)\}$ A partir de este sistema obtenemos un s.r.t R orientando de manera "adecuada" las ecuaciones:

1. $e \cdot x \to x$
2. $i(x) \cdot x \to e$
3. $(x \cdot y) \cdot z \to x \cdot (y \cdot z)$

Este sistema no es Church-Rosser pues el término $i(x) \cdot x$ se unifica con el subtérmino no variable $x \cdot y$ del lado izquierdo de la regla 3, produciéndose el término $(i(x) \cdot x) \cdot z$. Este término nos lleva al par crítico $(e \cdot z,\ i(x) \cdot (x \cdot z))$ que es un par no convergente, puesto que $i(x) \cdot (x \cdot z)$ es una forma normal y $e \cdot z$ solamente se reescribe en la forma normal z. Esto nos lleva a adoptar una nueva regla

4. $i(x) \cdot (x \cdot z) \to z$

Ahora tenemos un traslapamiento de las reglas 2 y 4 (el subíndice indica que regla se utilizó): $i(i(y)) \cdot (i(y) \cdot y) \to_4 y$, $i(i(y)) \cdot (i(y) \cdot y) \to_2 i(i(y)) \cdot e$, lo cual produce el par crítico $(y, i(i(y)) \cdot e)$ que no converge. Adoptamos la regla:

5. $i(i(y)) \cdot e \to y$

esta regla sera eliminada posteriormente, continuando con el análisis de la regla 4, tenemos un traslapamiento de 4 y 1: $i(e) \cdot (e \cdot z) \to_4 z$, $i(e) \cdot (e \cdot z) \to_1 i(e) \cdot z$, lo cual nos lleva a la nueva regla:

6. $i(e) \cdot z \to z$

Traslapamiento de 3 y 5: $(i(i(y)) \cdot e) \cdot x \to_3 i(i(y)) \cdot (e \cdot x)$, $(i(i(y)) \cdot e) \cdot x \to_5 y \cdot x$. Adoptamos la regla:

7. $i(i(y)) \cdot x \to y \cdot x$

Traslapamiento de 5 y 7: $i(i(y)) \cdot e \to_7 y \cdot e$, $i(i(y)) \cdot e \to_5 y$. Agregamos la regla

8. $y \cdot e \to y$

Traslapamiento de 5 y 8: $i(i(y)) \cdot e \to_5 y$, $i(i(y)) \cdot e \to_8 i(i(y))$. Agregamos la regla

9. $i(i(y)) \to y$

con la regla 9 eliminamos la regla 5 pues $i(i(y)) \cdot e \to_9 y \cdot e \to_8 y$ y la regla 7 ya que $i(i(y)) \cdot x \to_9 y \cdot x$.
Traslapamiento de 6 y 8: $i(e) \cdot e \to_6 e$, $i(e) \cdot e \to_8 i(e)$. Agregamos la regla

10. $i(e) \to e$

esta regla cancela a la 6, pues $i(e) \cdot z \to_{10} e \cdot z \to_1 z$.
Traslapamiento de 2 y 9: $i(i(y)) \cdot i(y) \to_2 e$, $i(i(y)) \cdot i(y) \to_9 y \cdot i(y)$, agregamos la regla

11. $y \cdot i(y) \to e$

Traslapamiento de 3 y 11: $(y \cdot i(y)) \cdot x \to_3 y \cdot (i(y) \cdot x)$, $(y \cdot i(y)) \cdot x \to_{11} e \cdot x$, adoptamos la regla

12. $y \cdot (i(y) \cdot x) \to x$

Tenemos otro traslapamiento de 3 y 11: $(x \cdot y) \cdot i(x \cdot y) \to_{11} e$, $(x \cdot y) \cdot i(x \cdot y) \to_3 x \cdot (y \cdot i(x \cdot y))$, agregamos la regla

13. $x \cdot (y \cdot i(x \cdot y)) \to e$

Traslapamiento de 13 y 4: $i(x) \cdot (x \cdot (y \cdot i(x \cdot y))) \to_4 y \cdot i(x \cdot y)$, $i(x) \cdot (x \cdot (y \cdot i(x \cdot y))) \to_{13} i(x) \cdot e$, adoptamos la regla

14. $y \cdot i(x \cdot y) \to i(x)$

esta regla cancela a la 13, pues $x \cdot (y \cdot i(x \cdot y)) \to_{14} x \cdot i(x) \to_{11} e$.
Por último, tenemos el traslapamiento de 4 y 14: $i(y) \cdot (y \cdot i(x \cdot y)) \to_4 i(x \cdot y)$, $i(y) \cdot (y \cdot i(x \cdot y)) \to_{14} i(y) \cdot i(x)$, agregamos la regla

15. $i(x \cdot y) \to i(y) \cdot i(x)$

esta regla cancela a la 14, ya que $y \cdot i(x \cdot y) \to_{15} y \cdot (i(y) \cdot i(x)) \to_{12} i(x)$.
Así hemos analizado todos los pares críticos y agregamos reglas de manera que todos convergieran. Con lo cual garantizamos que el sistema R' que consiste de:

1. $e \cdot x \to x$
2. $i(x) \cdot x \to e$
3. $(x \cdot y) \cdot z \to x \cdot (y \cdot z)$
4. $i(x) \cdot (x \cdot z) \to z$

8. $y \cdot e \to y$

9. $i(i(y)) \to y$

10. $i(e) \to e$

11. $y \cdot i(y) \to e$

12. $y \cdot (i(y) \cdot x) \to x$

15. $i(x \cdot y) \to i(y) \cdot i(x)$

es un s.r.t. terminal y confluente. El proceso de análisis de los pares críticos que realizamos fue bastante ineficiente, pues solamente nos guiamos por la intuición al momento de orientar las reglas; aunque en algunos casos la orientación es clara, en casos como el de la regla 15, la otra orientación también es plausible. No obstante, esto habría causado complicaciones que se describen en [KB70]. Este problema se resuelve mediante la elección de un orden de terminación para los términos que debemos proporcionarle al proceso de Knuth-Bendix como entrada. Sin embargo la manera de como encontrar tal orden sigue siendo una cuestión experimental.

Ahora presentamos una versión simple del proceso de compleción de Knuth-Bendix.

Precondición: Un sistema ecuacional E. Un orden de terminación $>$ para $TERM$.
$R := \varnothing$
while $E \neq \varnothing$ **do**
  Escoger una ecuación $s = t \in E$;
  Reducir s y t a formas normales s', t' con respecto a R
  if $s' \equiv t'$ **then**
    $E := E \setminus \{s = t\}$
  else
    if $s' > t'$ **then**
      $\alpha := s'$, $\beta := t'$
    else
      if $t' > s'$ **then**
        $\alpha := t'$, $\beta := s'$
      else
        falla
      end if
    end if
    $CP := \{p = q \mid (p, q)$ es un par crítico para $R \cup \{\alpha \to \beta\}\}$
    $R := R \cup \{\alpha \to \beta\}$;
    $E := (E \cup CP) \setminus \{s = t\}$
  end if
end while

éxito.

Postcondición: Un s.r.t. canónico R tal que $E_R \vdash s = t$ syss $E \vdash s = t$

A la hora de ejecutar este proceso tenemos tres posibilidades:

1. Terminación exitosa, en cuyo caso se obtiene un s.r.t. canónico.
2. Entrada a un ciclo infinito, provocando la no terminación.
3. Terminación con falla dado que un par de términos no pudo orientarse, es decir, ni $s > t$ ni $t > s$.

El tercer caso muestra la restricción más importante en el proceso, los sistemas ecuacionales que incluyan operadores conmutativos no pueden completarse. La prueba de que se obtiene un s.r.t. canónico en el caso 1 no es sencilla y queda fuera de esta exposición, el interesado puede consultarla en [Hue81]. Existe toda una rama dedicada al estudio del método de Knuth-Bendix de manera abstracta, para una introducción véase [BDP89, Der89].

En este capítulo hemos descrito algunos procesos de decisión, para el caso en que esté involucrada la igualdad, de manera que se puedan explotar sus propiedades características. El primer proceso, que involucra paramodulación con RFP-resolución, es de utilidad cuando tenemos cláusulas que contienen igualdades además de predicados ordinarios. El segundo proceso, que involucra un sistema de reescritura de términos canónico, es útil cuando se tienen puras igualdades.
Los capítulos 2 y 3 proporcionan una visión general de los procesos de decisión involucrados en OTTER además de que los teoremas de completud correspondientes justifican el funcionamiento correcto de dichos procesos.

En el siguiente capítulo presentamos la técnica de subsunción que proporciona un proceso de eliminación de cláusulas redundantes.

Capítulo 4

Subsunción

En este capítulo estudiaremos la técnica de *subsunción*[1] que es un proceso para eliminar cláusulas duplicadas o menos generales, es decir, cláusulas lógicamente más débiles que otras ya disponibles para el programa de razonamiento automático. El proceso de subsunción se debe a J. Alan Robinson (ver [Rob65]) y es considerado por algunos como su aportación más importante al campo del razonamiento automático, incluso por encima de la regla de resolución (consúltese [WOL91]).

4.1 Preliminares

Definición 4.1 Sea C una cláusula. Con $LIT(C)$ denotamos al conjunto de literales de C. Es decir, si $C = L_1 \vee \ldots \vee L_n$ entonces $LIT(C) = \{L_1, \ldots, L_n\}$

Definición 4.2 Sean C, D cláusulas. C *subsume* a D si existe una sustitución σ tal que $LIT(C)\sigma \subseteq LIT(D)$. Tal situación se denota con $C \leq_{ss} D$.
Si $C \leq_{ss} D$ y $D \leq_{ss} C$ entonces escribimos $C =_{ss} D$.

Ejemplo 4.1 Sean $C_1 = Px \vee Pfx$, $C_2 = Pfy \vee Pffy \vee Ry$, $C_3 = Pa \vee Pffa$, $C_4 = Pfy \vee Py \vee Pfz$. Tenemos las siguientes relaciones:

$$C_1 \leq_{ss} C_2 \quad \text{pues} \quad \{Px, Pfx\}\{x/fy\} \subseteq \{Pfy, Pffy, Ry\}$$

$$C_1 \leq_{ss} C_4 \quad \text{pues} \quad \{Px, Pfx\}\{x/y\} \subseteq \{Pfy, Py, Pfz\}$$

$$C_4 \leq_{ss} C_1 \quad \text{pues} \quad \{Pfy, Py, Pfz\}\{y/x, z/x\} \subseteq \{Px, Pfx\}$$

Obsérvese que tenemos $C_1 =_{ss} C_4$; además $C_1 \not\leq_{ss} C_3$ puesto que si existiera σ tal que $\{Px, Pfx\}\sigma \subseteq \{Pa, Pffa\}$ entonces hay dos casos:

$Px\sigma = Pa$ porque $Pfx\sigma = Pfa \notin LIT(C_3)$

$Px\sigma = Pffa$ por lo que $Pfx\sigma = Pffffa \notin LIT(C_3)$

[1] Del ingles *subsumption*, cuya traducción es subsunción o premisa menor

aunque $C_1 \not\leq_{ss} C_3$, obsérvese que $Px \leq_{ss} C_3$ y $Pfx \leq_{ss} C_3$.

A continuación mostramos algunas propiedades de la relación $\leq_{ss}$.

Proposición 4.1 *La relación $\leq_{ss}$ tiene las siguientes propiedades:*

1. *$\leq_{ss}$ es reflexiva.*
2. *$\leq_{ss}$ es transitiva.*
3. *Si $C \leq_{ss} D$ entonces $C \leq_{ss} D\vartheta$ para cualquier sustitución ϑ.*
4. *Si $C \leq_{ss} D$ entonces $\models \forall C \rightarrow \forall D$.*

dem:

1. Es inmediato utilizando la sustitución vacía.
2. Si $C \leq_{ss} D$ y $D \leq_{ss} E$ entonces se tienen las siguientes inclusiones. $LIT(C)\sigma \subseteq LIT(D)$ y $LIT(D)\mu \subseteq LIT(E)$. Así que $LIT(C)\sigma\mu \subseteq LIT(E)$. Es decir, $C \leq_{ss} E$.
3. Si $LIT(C)\sigma \subseteq LIT(D)$ entonces $LIT(C)\sigma\vartheta \subseteq LIT(D)\vartheta$, pero $LIT(D)\vartheta = LIT(D\vartheta)$. Por lo tanto $C \leq_{ss} D\vartheta$
4. Resulta inmediato puesto que $\models \forall C \rightarrow \forall C\sigma$ para cualquier σ y como existe alguna σ tal que $LIT(C)\sigma \subseteq LIT(D)$ entonces $\models \forall C\sigma \rightarrow \forall D$ para tal σ. Finalmente por transitividad obtenemos $\models \forall C \rightarrow \forall D$

⊣

La siguiente proposición será de utilidad para decidir el proceso de subsunción, pues asegura que para verificar si la cláusula C subsume a D, basta verificar si C subsume a cierta instancia cerrada de D.

Proposición 4.2 *Sean C, D cláusulas tales que $V(D) = \{x_1, \ldots, x_m\}$; $c_1, \ldots, c_m$ símbolos de constante que no figuran en C ni en D y $\sigma = \{x_1/c_1, \ldots, x_m/c_m\}$. Entonces $C \leq_{ss} D$ si y sólo si $C \leq_{ss} D\sigma$.*

dem:

$\Rightarrow$) Es inmediato por la proposición 4.1(3).

$\Leftarrow$) Supongamos que $C \leq_{ss} D\sigma$. Sea $\gamma = \{y_1/t_1, \ldots, y_n/t_n\}$ tal que $LIT(C)\gamma \subseteq LIT(D\sigma)$ y $V(C) = \{y_1, \ldots, y_n\}$. Como $D\sigma$ es cerrada entonces γ debe reemplazar todas las variables de C. Sean $s_i = t_i[c_1/x_1, \ldots, c_m/x_m]$, donde s/t indica el reemplazo de cada presencia del término s por t y $\mu = \{y_1/s_1, \ldots, y_n/s_n\}$. Obsérvese que $\mu = \gamma[c_1/x_1, \ldots, c_m/x_m]$. De manera que si $L \in LIT(C)$ entonces $L\mu = L\gamma[c_1/x_1, \ldots, c_m/x_m] \in LIT(D\sigma)[c_1/x_1, \ldots, c_m/x_m]$. Pero

nótese que $LIT(D\sigma)[c_1/x_1,\ldots,c_m/x_m] = LIT(D\sigma[c_1/x_1,\ldots,c_m/x_m])$ y además $D\sigma[c_1/x_1\ldots,c_m/x_m] = D$, porque las constantes c_i no figuraban en D. Por lo tanto concluimos que $L\mu \in LIT(D)$. Es decir, $LIT(C)\mu \subseteq LIT(D)$. Es decir, $C \leq_{ss} D$. ⊣

Con este resultado podemos decidir el proceso de subsunción.

Proposición 4.3 *El proceso de subsunción es decidible.*

dem:

Dadas dos cláusulas C y D queremos verificar si se cumple $C \leq_{ss} D$. Sin perder generalidad podemos suponer que la cláusula D siempre es cerrada. Sean $d = \tau(D)$ la profundidad[2] máxima de un término que figure en D y $\mathcal{H}$ el universo de Herbrand de la signatura de D. Definimos

$$\Lambda = \{\vartheta \mid \vartheta \text{ es una sustitución, } dom(\vartheta) = V(C),\ ran(\vartheta) \subseteq \mathcal{H}\}$$

De ésta definición resulta inmediato que $C \leq_{ss} D$ syss existe $\vartheta \in \Lambda$ tal que $LIT(C)\vartheta \subseteq LIT(D)$. Además obsérvese que si $\vartheta \in \Lambda$ y $\tau(ran(\vartheta)) > d$ entonces $\tau(C\vartheta) > d$ y por lo tanto $LIT(C\vartheta) \not\subseteq LIT(D)$.
Sea $\Lambda_d = \{\vartheta \in \Lambda \mid \tau(ran(\vartheta)) \leq d\}$, por la última observación tenemos que $C \leq_{ss} D$ syss existe $\vartheta \in \Lambda_d$ tal que $LIT(C)\vartheta \subseteq LIT(D)$. Ahora bien, como Λ_d es un conjunto finito entonces el proceso de decisión se obtiene al checar si $LIT(C)\vartheta \subseteq LIT(D)$ para $\vartheta \in \Lambda_d$. ⊣

Es claro que el proceso de decisión proporcionado es ineficiente y poco práctico, más adelante proporcionaremos un algoritmo de subsunción más efectivo.

Ahora vamos a extender el orden $\leq_{ss}$ a conjuntos de cláusulas de la manera usual.

Definición 4.3 Sean $\mathbb{C}, \mathbb{D}$ conjuntos de cláusulas. Entonces $\mathbb{C} \leq_{ss} \mathbb{D}$ syss para toda $D \in \mathbb{D}$ existe $C \in \mathbb{C}$ tal que $C \leq_{ss} D$.

La siguiente proposición muestra que la omisión de cláusulas subsumidas, al probar la satisfacibilidad de un conjunto, es correcta. Esto justifica el uso más simple del proceso de subsunción, que consiste en simplificar un conjunto de cláusulas mediante subsunción antes de proporcionarlo a un programa de razonamiento automático.

Proposición 4.4 *Sean* $\mathbb{C}$ *un conjunto de cláusulas y* $C \in \mathbb{C}$ *tal que* $\mathbb{C}-\{C\} \leq_{ss} \{C\}$. *Entonces*

$$\bigwedge_{D\in\mathbb{C}-\{C\}} \forall D \equiv \bigwedge_{D\in\mathbb{C}} \forall D$$

[2]La profundidad $\tau(t)$ del término t se define como 0 si t es un término simple y como $1 + \max\{\tau(t_i) \mid 1 \leq i \leq n\}$ si t es el término compuesto $f(t_1,\ldots,t_n)$. La profundidad de un conjunto de términos es la profundidad máxima de un término que figure en él.

dem:

$\Leftarrow$) Es trivial.

$\Rightarrow$) Supongamos que $\bigwedge_{D\in\mathbb{C}-\{C\}} \forall D$.
Por hipótesis existe $D \in \mathbb{C} - \{C\}$ tal que $D \leq_{ss} C$. Además sabemos que $\models \forall D \rightarrow \forall C$, por la proposición 4.1(4), y por nuestra suposición tenemos $\forall D$. Por lo tanto obtenemos $\forall C$, lo cual junto con la suposición nos lleva a concluir que $\bigwedge_{D\in\mathbb{C}} \forall D$. ⊣

Así que en adelante podemos considerar solamente conjuntos de cláusulas reducidos bajo subsunción, es decir, conjuntos $\mathbb{C}$ tales que para cualesquiera $C_1, C_2 \in \mathbb{C}$, si $C_1 \leq_{ss} C_2$ entonces $C_1 = C_2$.

Más importante y complicado, desde el punto de vista computacional, que el uso de la subsunción como proceso preliminar es su uso durante la derivación de cláusulas. Al combinar subsunción con resolución o con algún refinamiento tenemos tres posibilidades:

- Subsunción hacia adelante (*forward subsumption*). Este proceso consiste en eliminar las cláusulas recién derivadas que son subsumidas por cláusulas derivadas anteriormente.
- Subsunción hacia atrás (*backward subsumption*). En este caso una cláusula derivada con anterioridad y que es subsumida por una cláusula recién obtenida se elimina. Esto provoca algunos problemas de completud como mencionaremos en la siguiente sección.
- Reemplazo. En este caso el conjunto de cláusulas derivadas se reduce periódicamente bajo subsunción, es decir, después de cierto periodo de tiempo fijo, se aplica la subsunción al conjunto.

4.2 Completud de Resolución bajo Subsunción

El uso de subsunción como un paso previo a la deducción no trae problemas de completud, e incluso ofrece ventajas, pues si $\mathbb{C}'$ se obtiene de $\mathbb{C}$ mediante subsunción, entonces $R_x(\mathbb{C}') \subseteq R_x(\mathbb{C})$ y $\square \in R_x(\mathbb{C}')$ syss $\square \in R_x(\mathbb{C})$.

El uso de subsunción durante el proceso de derivación depende esencialmente del método de búsqueda. Como ya se dijo, la subsunción hacia atrás provoca problemas de completud, ya que no podemos eliminar la cláusula C obtenida anteriormente sin afectar el árbol de derivación. Para evitar esto lo que suele hacerse es desactivar la cláusula C en lugar de eliminarla de manera que si al final la cláusula C' que subsumió a C no resulta ser ancestro de C, entonces ésta se activa nuevamente.

Por otra parte, la combinación de subsunción hacia adelante con un operador de refinamiento siempre define un refinamiento.

Definición 4.4 Sean $\mathbb{C}, \mathbb{D}$ conjuntos de cláusulas y $nsa(\mathbb{C}, \mathbb{D}) = \{D \in \mathbb{D} \mid \mathbb{C} \not\leq_{ss} \{D\}\}$ el conjunto de cláusulas de $\mathbb{D}$ que no son subsumidas por $\mathbb{C}$. Sea R_x un operador de refinamiento de resolución. Definimos

$$\begin{aligned} S^0_{xs}(\mathbb{C}) &= S^0_x(\mathbb{C}) \\ S^{n+1}_{xs}(\mathbb{C}) &= S^n_{xs}(\mathbb{C}) \cup nsa(S^n_{xs}(\mathbb{C}), \rho_x(S^n_{xs}(\mathbb{C}))) \\ R_{xs}(\mathcal{S}) &= \bigcup_{n \in \mathbf{N}} S^n_{xs}(\mathbb{C}) \end{aligned}$$

R_{xs} es el *operador de refinamiento R_x bajo subsunción hacia adelante.*

Los siguientes lemas constituyen el camino hacia el teorema de completud de resolución bajo subsunción hacia adelante.

Lema 4.1 *La subsunción se preserva bajo factores, es decir, si C, D son cláusulas tales que $C \leq_{ss} D$ y D' es un factor de D entonces $C \leq_{ss} D'$.*

dem:

Esta demostración es nuestra respuesta a un ejercicio de [Lei97].
Mediante inducción sobre el grado del factor D'.

- Base de la inducción. D' es un factor trivial de D. En tal caso existe una sustitución σ tal que $D\sigma = D'$, y como $C \leq_{ss} D$ entonces $C \leq_{ss} D\sigma = D'$ por la proposición 4.1(3).
- Hipótesis de Inducción: Si D' es un k-factor de D y $C \leq_{ss} D$ entonces $C \leq_{ss} D'$.
- Sea D' un $(k+1)$-factor de D. En tal caso podemos obtener un k-factor D'' de D tal que D' es un 1-factor de D''. Más aún, D' es un reducto de D''.
  Por HI tenemos que $C \leq_{ss} D''$, es decir, $LIT(C)\eta \subseteq LIT(D'')$ para alguna sustitución η. Pero como D' es un reducto de D'' entonces $LIT(D'') = LIT(D')$. Por lo tanto $LIT(C)\eta \subseteq LIT(D')$, es decir, $C \leq_{ss} D'$.

⊣

Lema 4.2 *Sean C_1, D_1, C_2, D_2 cláusulas tales que $C_1 \leq_{ss} D_1, C_2 \leq_{ss} D_2$. Si D es un resolvente de D_1 y D_2 entonces se cumple alguna de las siguientes propiedades:*

a) $C_1 \leq_{ss} D$.

b) $C_2 \leq_{ss} D$.

c) Existe un resolvente C de C_1 y C_2 tal que $C \leq_{ss} D$.

dem:

Por la definición 2.7 existen factores con variables ajenas, $E_1 \vee M \vee F_1$ de D_1 y $E_2 \vee N \vee F_2$ de D_2 tales que $\{M, N^c\}$ es unificable mediante el umg σ y $D = (E_1 \vee F_1 \vee E_2 \vee F_2)\sigma$. Como la subsunción se preserva bajo factores (lema 4.1) entonces $C_1 \leq_{ss} E_1 \vee M \vee F_1$ y $C_2 \leq_{ss} E_2 \vee N \vee F_2$, es decir, existen ϑ_1, ϑ_2 tales que $LIT(C_1)\vartheta_1 \subseteq LIT(E_1 \vee M \vee F_1)$ y $LIT(C_2)\vartheta_2 \subseteq LIT(E_2 \vee N \vee F_2)$. Vamos a analizar dos casos

- $M \notin LIT(C_1)\vartheta_1$ o $N \notin LIT(C_2)\vartheta_2$.
  Supongamos que $M \notin LIT(C_1)\vartheta_1$. En tal caso $LIT(C_1)\vartheta_1 \subseteq LIT(E_1 \vee F_1)$, es decir, $C_1 \leq_{ss} E_1 \vee F_1$. Además tenemos $E_1 \vee F_1 \leq_{ss} (E_1 \vee F_1)\sigma \leq_{ss} (E_1 \vee F_1 \vee E_2 \vee F_2)\sigma = D$. Por lo tanto $C_1 \leq D$, de manera que se cumple la condición a).

  En el caso en el que $N \notin LIT(C_2)\vartheta_2$ se prueba análogamente que $C_2 \leq_{ss} D$, es decir, se cumple la condición b).

- $M \in LIT(C_1)\vartheta_1$ y $N \in LIT(C_2)\vartheta_2$.
  Sean $\mathcal{L}_1 = \{L \in LIT(C_1) \mid L\vartheta_1 = M\}$ y $\mathcal{L}_2 = \{L \in LIT(C_2) \mid L\vartheta_2 = N\}$. Obsérvese que ϑ_i unifica a $\mathcal{L}_i$, por lo que existen unificadores más generales λ_1, λ_2 de $\mathcal{L}_1, \mathcal{L}_2$ respectivamente. En tal situación, las cláusulas $C_1\lambda_1$, $C_2\lambda_2$ son G-instancias de C_1, C_2.
  Más aún existen factores irreducibles $G_1 \vee R \vee H_1$ de C_1 y $G_2 \vee S \vee H_2$ de C_2 tales que $R \leq_s M$, $G_1 \vee H_1 \leq_{ss} E_1 \vee F_1$, $S \leq_s N$, $G_2 \vee H_2 \leq_{ss} E_2 \vee F_2$. Estos se obtienen tomando el mayor reducto posible de $C_1\lambda_1$ y $C_2\lambda_2$ respectivamente.
  Sean η_1, η_2 tales que $\lambda_1\eta_1 = \vartheta_1, \lambda_2\eta_2 = \vartheta_2$, entonces $R\eta_1 = M$, $S\eta_2 = N$, $LIT(G_1 \vee H_1)\eta_1 \subseteq LIT(E_1 \vee F_1)$, $LIT(G_2 \vee H_2)\eta_2 \subseteq LIT(E_2 \vee F_2)$. Definimos $\eta = \eta_1 \cup \eta_2$, lo cual es correcto pues $V(E_1 \vee M \vee F_1) \cap V(E_2 \vee N \vee F_2) = \varnothing$.
  Obsérvese que $R\eta = M$ y $S\eta = N$, por lo tanto el conjunto $\{R, S^c\}$ es unificable mediante $\eta\sigma$ y podemos encontrar un umg de $\{R, S^c\}$, digamos τ. Por lo tanto existe ρ tal que $\tau\rho = \eta\sigma$. Como $LIT(G_1 \vee H_1)\eta_1 \subseteq LIT(E_1 \vee F_1)$ entonces $LIT(G_1 \vee H_1)\eta\sigma \subseteq LIT(E_1 \vee F_1)\sigma$, es decir, $LIT(G_1 \vee H_1)\tau\rho \subseteq LIT(E_1 \vee F_1)\sigma$. Análogamente concluimos que $LIT(G_2 \vee H_2)\tau\rho \subseteq LIT(E_2 \vee F_2)\sigma$.
  Finalmente observemos que la cláusula $C = (G_1 \vee H_1 \vee G_2 \vee H_2)\tau$ es un resolvente de C_1 y C_2 y que $LIT(C)\rho = LIT(G_1 \vee H_1 \vee G_2 \vee H_2)\tau\rho \subseteq LIT(E_1 \vee F_1 \vee E_2 \vee F_2)\sigma = LIT(D)$. Por lo tanto $C \leq_{ss} D$, es decir, se cumple la condición c).

⊣

El siguiente lema, junto con el teorema de completud son una adaptación nuestra, basada en un teorema más general de [Lei97].

Lema 4.3 *Sean* $\mathbb{C}$ *un conjunto de cláusulas y* $R_{\varnothing}$ *el operador de resolución ilimitada. Entonces las cláusulas generadas por* $R_{\varnothing s}$ *subsumen a las generadas por* $R_{\varnothing}$. *Es decir,* $R_{\varnothing s}(\mathbb{C}) \leq_{ss} R_{\varnothing}(\mathbb{C})$.

dem:

Basta ver que para toda n, las cláusulas del estrato n de $R_{\varnothing s}$ subsumen a las cláusulas de $R_{\varnothing}$. Es decir, $S^n_{\varnothing s}(\mathbb{C}) \leq_{ss} S^n_{\varnothing}(\mathbb{C})$. Esto lo haremos mediante inducción sobre n.

- $n = 0$. Es trivial.
- HI. $S^n_{\varnothing s}(\mathbb{C}) \leq_{ss} S^n_{\varnothing}(\mathbb{C})$.
- P.D. $S^{n+1}_{\varnothing s}(\mathbb{C}) \leq_{ss} S^{n+1}_{\varnothing}(\mathbb{C})$, es decir,

  $$S^n_{\varnothing s}(\mathbb{C}) \cup nsa(S^n_{\varnothing s}(\mathbb{C}), \varphi(Res(S^n_{\varnothing s}(\mathbb{C})))) \leq_{ss} S^n_{\varnothing}(\mathbb{C}) \cup \varphi(Res(S^n_{\varnothing}(\mathbb{C})))$$

  Sea $D \in S^n_{\varnothing}(\mathbb{C}) \cup \varphi(Res(S^n_{\varnothing}(\mathbb{C})))$.
  Basta ver el caso en que $D \in \varphi(Res(S^n_{\varnothing}(\mathbb{C})))$ ya que el otro caso resulta inmediato de la HI.
  Sea $D \in \varphi(Res(S^n_{\varnothing}(\mathbb{C})))$. Así que existen $D_1, D_2 \in S^n_{\varnothing}(\mathbb{C})$ tales que D es un resolvente de D_1 y D_2. Mediante la HI, tomemos $C_1, C_2 \in S^n_{\varnothing s}(\mathbb{C})$ tales que $C_1 \leq_{ss} D_1$ y $C_2 \leq_{ss} D_2$. Tenemos los siguientes casos, de acuerdo al lema 4.2.

  1. $C_1 \leq_{ss} D$. En cuyo caso, como $C_1 \in S^n_{\varnothing s}(\mathbb{C})$, podemos concluir que $S^n_{\varnothing s}(\mathbb{C}) \leq_{ss} \{D\}$ y por lo tanto $S^{n+1}_{\varnothing s}(\mathbb{C}) \leq_{ss} \{D\}$
  2. $C_2 \leq_{ss} D$. Análogamente al caso anterior obtenemos $S^{n+1}_{\varnothing s}(\mathbb{C}) \leq_{ss} \{D\}$.
  3. Existe C resolvente de C_1, C_2 tal que $C \leq_{ss} D$. Aquí hay dos subcasos.
     (a) $S^n_{\varnothing s}(\mathbb{C}) \leq_{ss} \{C\}$. En cuyo caso, por la transitividad de $\leq_{ss}$ obtenemos $S^n_{\varnothing s}(\mathbb{C}) \leq_{ss} \{D\}$, de donde concluimos que $S^{n+1}_{\varnothing s}(\mathbb{C}) \leq_{ss} \{D\}$
     (b) $S^n_{\varnothing s}(\mathbb{C}) \not\leq_{ss} \{C\}$. Dado que $C \in Res(S^n_{\varnothing s}(\mathbb{C}))$ entonces concluimos que $C \in nsa(S^n_{\varnothing s}(\mathbb{C}), \varphi(Res(S^n_{\varnothing s}(\mathbb{C}))))$. Por lo tanto, como $C \leq_{ss} D$, obtenemos que $S^{n+1}_{\varnothing s}(\mathbb{C}) \leq_{ss} \{D\}$.

De manera que en cualquier caso obtenemos que $S^{n+1}_{\varnothing s}(\mathbb{C}) \leq_{ss} \{D\}$ y como $D \in S^{n+1}_{\varnothing}(\mathbb{C})$, *entonces concluimos que* $S^{n+1}_{\varnothing s}(\mathbb{C}) \leq_{ss} S^{n+1}_{\varnothing}(\mathbb{C})$. ⊣

Ahora podemos demostrar el teorema de completud correspondiente.

Teorema 4.1 *(Completud de Resolución bajo Subsunción hacia adelante) Si* $\mathbb{C}$ *es un conjunto no satisfacible de cláusulas entonces* $\square \in R_{\varnothing s}(\mathbb{C})$.

dem:

Como $R_{\varnothing}$ es completo entonces $\square \in R_{\varnothing}(\mathbb{C})$ y por el lema 4.3 tenemos que $R_{\varnothing s}(\mathbb{C}) \leq_{ss} R_{\varnothing}(\mathbb{C})$, por lo que existe $C \in R_{\varnothing s}(\mathbb{C})$ tal que $R \leq_{ss} \square$. Pero esto solo es posible si $R = \square$. Por lo tanto $\square \in R_{\varnothing s}(\mathbb{C})$. ⊣

4.3 Completud de Hiperresolución bajo Reemplazo

El proceso de reemplazo consiste en eliminar cada cláusula subsumida sin importar como ni cuando fue derivada. Mientras que los refinamientos son un método más cercano a las derivaciones, el reemplazo es esencialmente un método de reducción.
Dado un conjunto de cláusulas $\mathbb{C}$, éste podría tener más de un conjunto equivalente y reducido bajo subsunción. Sin embargo, si dotamos al conjunto con un orden (como el lexicográfico) y mantenemos la cláusula más pequeña de cada clase de equivalencia según $=_{ss}$ obtenemos un conjunto único al reducir bajo subsunción a $\mathbb{C}$.
De esta manera podemos considerar la reducción bajo subsunción como un operador $sub(\mathbb{C})$ que tiene las siguientes propiedades:

1. $sub(\mathbb{C}) \subseteq \mathbb{C}$.
2. $sub(sub(\mathbb{C})) = sub(\mathbb{C})$.
3. Para cualquier $\mathbb{D}$, si $\mathbb{C} \leq_{ss} \mathbb{D}$ entonces $sub(\mathbb{C} \cup \mathbb{D}) =_{ss} sub(\mathbb{C})$.

Con ayuda de este operador definimos el concepto correspondiente a los operadores de refinamiento.

Definición 4.5 Sean R_x un operador de refinamiento y $\mathbb{C}$ un conjunto de cláusulas. Definimos

$$S^0_{xr}(\mathbb{C}) = sub(\mathbb{C})$$

$$S^{n+1}_{xr}(\mathbb{C}) = sub(S^n_{xr}(\mathbb{C}) \cup \rho_x(S^n_{xr}(\mathbb{C})))$$

La sucesión $\langle S^n_{xr}(\mathbb{C})\rangle_{n\in\mathbb{N}}$ es la *R_x-sucesión de reemplazo* o *R_{xr}-sucesión* para $\mathbb{C}$.

Definición 4.6 Una R_x-sucesión de reemplazo $\langle S^n_{xr}(\mathbb{C})\rangle_{n\in\mathbb{N}}$ es *convergente* si existe $k \in \mathbb{N}$ tal que para toda $l \geq k$, $S^l_{xr}(\mathbb{C}) = S^k_{xr}(\mathbb{C})$. En otro caso la sucesión es *divergente*.

Definición 4.7 Una R_x-sucesión de reemplazo $\langle S^n_{xr}(\mathbb{C})\rangle_{n\in\mathbb{N}}$ es *refutacional* si existe $k \in \mathbb{N}$ tal que $S^k_{xr}(\mathbb{C}) = \{\square\}$.

Ejemplo 4.2 Sea $\mathbb{C} = \{Px_1 \vee Qx_1, Px_1 \vee \neg Qx_1, Qx_1 \vee \neg Px_1, \neg Px_1 \vee \neg Qx_1\}$. Entonces

1. La sucesión
$$\mathbb{C}, \{Px_1, Qx_1, \neg Px_1, \neg Qx_1\}, \{\square\}$$
es una $R_{\varnothing_T}$-sucesión refutacional.

2. La sucesión
$$\mathbb{C}, \{Px_1, Qx_1\}, \{\varnothing\}$$
es una R_{H_T}-sucesión refutacional.

3. Si $\mathbb{D} = \mathbb{C} - \{\neg Px_1 \vee \neg Qx_1\}$ entonces $\mathbb{D}$ es satisfacible y la sucesión
$$\mathbb{D}, \{Px_1, Qx_1\}, \{Px_1, Qx_1\}, \ldots$$
es una R_{H_T}-sucesión convergente.

4. Si $\mathbb{D} = \{Pa, Pfx_1 \vee \neg Px_1\}$ entonces $\mathbb{D}$ es satisfacible y la sucesión de reemplazo para $\mathbb{D}$ es:
$$\mathbb{D}, \mathbb{D} \cup \{Pfa\}, \mathbb{D} \cup \{Pfa, Pf^2a\}, \ldots$$
que resulta divergente pues $Pf^ka \in S^k_{H_T}(\mathbb{C}) - S^{k-1}_{H_T}(\mathbb{C})$ para $k > 0$.

5. Si $\mathbb{D} = \{Pf^2x_1, Pfx_1 \vee \neg Px_1\}$ entonces la sucesión de reemplazo para $\mathbb{D}$ converge. De hecho $S^1_{H_T}(\mathbb{D}) = S^0_{H_T}(\mathbb{D}) = \mathbb{D}$ pues $\{Pf^3x_1\} = \rho_H(\mathbb{D})$ y $Pf^2x_1 \leq_{ss} Pf^3x_1$.

Como se observa en el ejemplo anterior, si un conjunto es satisfacible, nada se puede decir acerca de la convergencia de la sucesión de reemplazo. Para los conjuntos no satisfacibles, como veremos, la sucesión de reemplazo es refutacional, y es en este sentido como entendemos la completud bajo reemplazo. Específicamente probaremos la completud de hiperresolución bajo reemplazo, la prueba se sirve en gran medida de los siguientes lemas.

Definición 4.8 Sean C, D cláusulas en PN-forma. Definimos la relación $\leq_{ss0}$ como: $C \leq_{ss0} D$ syss existe una sustitución ϑ tal que $LIT(C^+)\vartheta \subseteq LIT(D^+)$ y $C^-\vartheta = D^-$

Resulta inmediato que si $C \leq_{ss0} D$ entonces $C \leq_{ss} D$. El siguiente lema mejora la exposición original de [Lei97], en sentido de claridad.

Lema 4.4 *Sean* $\Gamma' : (C; D'_1, \ldots D'_n), \Gamma : (C; D_1, \ldots D_n)$ *sucesiones conflictivas tales que* $D_i \leq_{ss} D'_i$. *Sea* R'_i *el i-ésimo resolvente intermedio de* Γ'. *Entonces existe una cláusula* R_i *tal que* $R_i \leq_{ss0} R'_i$.

dem:

Vamos a construir R_i recursivamente. Para $i = 0$ tenemos $R'_0 = C$ y definimos $R_0 = R'_0$. Trivialmente $R_0 \leq_{ss0} R'_0$.
Supongamos que R_i se ha definido, para $i < n$. Hay que analizar dos casos.

a) R_i es positiva. En tal caso definimos $R_{i+1} = R_i$. Como $R_i \leq_{ss0} R'_i$ y R_i es positiva entonces $R_i \leq_{ss} R'_i$. Dado que R'_{i+1} es un RFP-resolvente de R'_i y D_i entonces R'_i debe ser de la forma $R'_i = E'_{i+1} \vee \neg Q'$ donde Q' es un átomo. Así que claramente $R_i \leq_{ss} E'_{i+1}$, pues R_i es positiva y $R_i \leq_{ss} R'_i$. Además R'_{i+1} contiene una instancia de E'_{i+1} pues es un RFP-resolvente de $E'_{i+1} \vee \neg Q'$ y D'_i y el átomo que se resuelve es Q'. De esto concluimos que $R_i \leq_{ss} R'_{i+1}$, es decir, $R_{i+1} \leq_{ss} R'_{i+1}$, pero como R_{i+1} es positiva entonces $R_{i+1} \leq_{ss0} R'_{i+1}$.

b) R_i no es positiva. Tenemos dos subcasos.

ba) $D_{i+1} \leq_{ss} R'_{i+1}$. En este caso definimos $R_{i+1} = D_{i+1}$ y como D_{i+1} es positiva entonces $R_{i+1} \leq_{ss0} R'_{i+1}$.

bb) $D_{i+1} \not\leq_{ss} R'_{i+1}$. Como $R_i \leq_{ss0} R'_i$ entonces existe una sustitución ϑ tal que $LIT(R_i^+)\vartheta \subseteq LIT(R_i'^+)$ y $R_i^-\vartheta = R_i'^-$. Puesto que R'_{i+1} es un RFP-resolvente de R'_i y D'_{i+1} podemos encontrar cláusulas negativas T_i, T'_i (o tal vez $T_i = T'_i = \square$) tales que $R_i = R_i^+ \vee T_i \vee \neg Q$, $R'_i = R_i'^+ \vee T'_i \vee \neg Q'$, $T_i\vartheta = T'_i$ y $Q\vartheta = Q'$.
Sea $F'_{i+1} = G'_1 \vee P' \vee G'_2$ un factor de D'_{i+1} tal que P' es la literal sobre la que se resolvió en el resolvente binario de R'_i y F'_{i+1}. Como por hipótesis $D_{i+1} \leq_{ss} D'_{i+1}$, hay dos posibilidades.

bba) $D_{i+1} \leq_{ss} G'_1 \vee G'_2$. Entonces claramente $D_{i+1} \leq_{ss} R'_{i+1}$ y concluimos como en el subcaso ba).

bbb) Existe un factor F_{i+1} de D_{i+1} tal que $F_{i+1} = G_1 \vee P \vee G_2$, donde $LIT(G_1 \vee G_2)\eta \subseteq LIT(G'_1 \vee G'_2)$ y $P\eta = P'$ para alguna sustitución η. Sea σ un umg de $\{P', Q'\}$, entonces $R'_{i+1} = (G'_1 \vee G'_2 \vee R_i'^+ \vee T'_i)\sigma$. Suponiendo spg que $dom(\eta) \cap dom(\vartheta) = \varnothing$, $\eta \cup \vartheta$ está bien definida y $(\eta \cup \vartheta)\sigma$ unifica a $\{P, Q\}$. Por lo tanto podemos tomar un umg τ de $\{P, Q\}$.
Definimos $R_{i+1} = (G_1 \vee G_2 \vee R_i^+ \vee T_i)\tau$ y como existe ρ tal que $\tau\rho = (\eta \cup \vartheta)\sigma$ es fácil ver que $R_{i+1} \leq_{ss0} R'_{1+1}$.

⊣

Lema 4.5 *Las sucesiones de reemplazo para hiperresolución subsumen a los estratos del operador de hiperresolución (sin subsunción). Es decir, si $\mathbb{C}$ es un conjunto de N_c-cláusulas entonces para toda $k \in \mathbb{N}$, $S^k_{Hr}(\mathbb{C}) \leq_{ss} S^k_H(\mathbb{C})$.*

dem:

El lector puede cerciorarse de que $S^k_H(sub(\mathbb{C})) = S^k_h(\mathbb{C})$ para toda $k \in \mathbb{N}$, por lo que podemos suponer s.p.g. que $\mathbb{C}$ es reducido bajo subsunción. Dicho lo cual procedemos por inducción sobre k.

- $k = 0$. Tenemos $S^0_H(\mathbb{C}) = \mathbb{C} = sub(\mathbb{C}) = S^0_{Hr}(\mathbb{C})$. De manera que $S^0_{Hr}(\mathbb{C}) \leq_{ss} S^0_H(\mathbb{C})$ trivialmente.

- Hipótesis de Inducción. $S^k_{Hr}(\mathbb{C}) \leq_{ss} S^k_H(\mathbb{C})$.

- P.D. $S^{k+1}_{Hr}(\mathbb{C}) \leq_{ss} S^{k+1}_H(\mathbb{C})$.
  Claramente $S^{k+1}_{Hr}(\mathbb{C}) \leq_{ss} S^k_{Hr}(\mathbb{C})$ por lo que, usando la HI y la transitividad de $\leq_{ss}$ tenemos $S^{k+1}_{Hr}(\mathbb{C}) \leq_{ss} S^k_H(\mathbb{C})$. Así que basta con demostrar que $S^{k+1}_{Hr}(\mathbb{C}) \leq_{ss} \rho_H(S^k_H(\mathbb{C}))$.
  Sea $E' \in \rho_H(S^k_H(\mathbb{C}))$, es decir, existe una sucesión conflictiva $\Gamma' : (C'; D'_1, \ldots, D'_n)$, con $C', D'_i \in S^k_H(\mathbb{C})$ tal que E' es un hiperresolvente de Γ'.
  Por HI existen $C, D_1, \ldots, D_n \in S^k_{Hr}(\mathbb{C})$ tales que $C \leq_{ss} C'$ y $D_i \leq_{ss} D'_i$.
  Si C no es positiva entonces $\Gamma : (C; D_1, \ldots, D_n)$ es una sucesión conflictiva sobre $S^k_{Hr}(\mathbb{C})$. Tenemos dos casos, el caso fácil es cuando $C \notin \mathbb{C}$.
  Como $C \in S^k_{Hr}(\mathbb{C}) - \mathbb{C}$ entonces $C = C^+$, es decir, C es positiva y dado que $C \leq_{ss} C'$ entonces existe ϑ tal que $LIT(C)\vartheta \subseteq LIT(C'^+)$. Puesto que C' es el nucleo de Γ' entonces E' debe contener una subcláusula de la forma $C'^+\eta$ para alguna sustitución η. De lo anterior concluimos que $LIT(C)\vartheta\eta \subseteq LIT(E')$, es decir, $C \leq_{ss} E'$. Como $C \in S^k_{Hr}(\mathbb{C})$ y $S^{k+1}_{Hr}(\mathbb{C}) \leq_{ss} S^k_{Hr}(\mathbb{C})$ entonces existe $D \in S^{k+1}_{Hr}(\mathbb{C})$ tal que $D \leq_{ss} C$. Por lo tanto $D \leq_{ss} E'$ y terminamos este caso.
  Veamos ahora qué sucede cuando $C \in \mathbb{C}$. Tenemos $C' \in \mathbb{C}$ pues en otro caso Γ' no sería una sucesión conflictiva y además como $\mathbb{C}$ es reducido bajo subsunción y $C \leq_{ss} C'$ entonces $C = C'$, en particular $\Gamma : (C; D_1, \ldots, D_n)$ es una sucesión conflictiva sobre $S^k_{Hr}(\mathbb{C})$. Aplicando el lema 4.4 con $\Gamma' : (C; D'_1, \ldots, D'_n)$ y $\Gamma : (C; D_1, \ldots, D_n)$, obtenemos que $R_n \leq_{ss0} R'_n = E'$. Por la manera en que se construyen los R_i en el lema 4.4, R_n es D_i para alguna $i \leq n$ o bien es hiperresolvente de una sucesión conflictiva $(C; D_1, \ldots, D_l)$ para alguna $l \leq n$. Por lo tanto $R_n \in S^k_{Hr}(\mathbb{C}) \cup \rho_H(S^k_{Hr}(\mathbb{C}))$. Si definimos $E = R_n$, entonces $E \leq_{ss} E'$, pues $E \leq_{ss0} E'$ y E es positiva; además existe una cláusula $E_0 \in sub(S^k_{Hr}(\mathbb{C}) \cup \rho_H(S^k_{Hr}(\mathbb{C})))$ tal que $E_0 \leq_{ss} E$, (E_0 podría ser E). Por lo tanto existe $E_0 \in S^{k+1}$ tal que $E_0 \leq_{ss} E$.
  Por lo tanto $S^{k+1}_{Hr}(\mathbb{C}) \leq_{ss} S^{k+1}_H(\mathbb{C})$ y la prueba está terminada.

⊣

Finalmente concluimos el teorema de completud correspondiente.

Teorema 4.2 *(Completud de Hiperresolución bajo Reemplazo).*
Sea $\mathbb{C}$ un conjunto no satisfacible de N_c-cláusulas. Entonces la sucesión de reemplazo para $\mathbb{C}$, $\langle S^n_{xr}(\mathbb{C})\rangle_{n\in\mathbb{N}}$ es una sucesión refutacional.

dem:

Como $\mathbb{C}$ es no satisfacible, el teorema 2.7 garantiza que existe $k \in \mathbb{N}$ tal que $\square \in S^k_H(\mathbb{C})$. Mediante el lema 4.5 sabemos que $S^k_{Hr}(\mathbb{C}) \leq_{ss} S^k_H(\mathbb{C})$ y como únicamente $\square$ subsume a $\square$, concluimos que $\square \in S^k_{Hr}(\mathbb{C})$ lo cual implica que $S^k_{Hr} = \{\square\}$, es decir, $\langle S^n_{xr}(\mathbb{C})\rangle_{n\in\mathbb{N}}$ es una sucesión refutacional. ⊣

4.4 El Algoritmo de Stillman para Subsunción

En la proposición 4.3 mostramos que el proceso de subsunción es decidible. No obstante, el procedimiento de decisión que puede obtenerse de esa proposición es altamente ineficaz. En esta sección presentamos un procedimiento de decisión óptimo mediante el algoritmo de Stillman [Sti73] para subsunción. La idea principal del algoritmo consiste en buscar la sustitución adecuada de izquierda a derecha. Veamos un ejemplo.

Ejemplo 4.3 Sean $C = Px \vee Pfxy$, $D = Pa \vee Pb \vee Pfba$. Queremos decidir si $C \leq_{ss} D$. Para enviar la primera literal Px de C a D tenemos varios candidatos, a saber, $\vartheta_1 = \{x/a\}$, $\vartheta_2 = \{x/b\}$, $\vartheta_3 = \{x/fba\}$. Claramente la elección adecuada es ϑ_2 que podemos extender a $\mu = \{x/b, y/a\}$ con lo cual tenemos $LIT(C)\mu \subseteq LIT(D)$, es decir, $C \leq_{ss} D$.
Si hubieramos elegido ϑ_1 eventualmente debemos corregir la elección con ϑ_2. Supóngase que intercambiamos las literales de C de modo que $Pfxy$ es la primera; entonces claramente μ es la única sustitución que permite la subsunción

Con este ejemplo se ve la importancia del orden de las literales en el proceso algorítmico para hallar una sustitución que subsuma.
La idea del algoritmo de Stillman, ST en adelante, es la siguiente: Sea $C = L_1 \vee \ldots \vee L_n$ y supóngase que ya se halló ϑ_i tal que $\{L_1 \vee \ldots \vee L_i\}\vartheta_i \subseteq LIT(D)$, si se logra extender alguna ϑ_i a ϑ_n entonces $C \leq_{ss} D$, en caso contrario $C \not\leq_{ss} D$. En el ejemplo anterior, ϑ_2 puede extenderse a μ lo cual no es posible con ϑ_1, ϑ_3.
Para la especificación de ST usamos las siguientes funciones auxiliares: $unif(L, M)$ que es una función booleana que es verdadera syss $\{L, M\}$ es unificable; la función $umg(L, M)$ que calcula un umg de $\{L, M\}$ en caso de que este conjunto sea unificable y está indefinida en otro caso. ST queda representado mediante la función booleana 5-aria $STI(C, D, i, j, \vartheta)$ donde i apunta a las literales de C, j apunta a las literales de D y ϑ es una sustitución. Esta versión del algoritmo es nuestra y se basa en la versión de [Lei97].
Precondición: $C = L_1 \vee \ldots \vee L_{|C|}$, $D = K_1 \vee \ldots \vee K_{|D|}$

```
function STI(C, D, i, j, ϑ):Boolean.
begin
if j >| C | then
  STI(C, D, i, j, ϑ) := false;
else
  a := j
  while not unif(L_i ϑ, K_a) and a ≤| D | do
    a := a + 1;
  end while
  if a >| D | then
    STI(C, D, i, j, ϑ) := false;
  else
    μ_i := umg(L_i ϑ, K_a);
    if i =| C | or STI(C, D, i + 1, 1, ϑμ_i) then
```

```
        STI(C, D, i, j, ϑ) := true;
      else
        STI(C, D, i, a + 1, ϑ)
      end if
    end if
  end if
  end
```

```
  begin
  ST(C, D) := STI(C, D, 1, 1, ∅)
  if ST(C, D) then
    write('C ≤ss D');
  else
    write('C ⋠ss D');
  end if
  end
```

Postcondición: $LIT(C)\vartheta_n \subseteq LIT(D)$ o $C \not\leq_{ss} D$.

La ejecución de ST puede describirse con ayuda de un árbol que definimos enseguida.

Definición 4.9 Sean $C = L_1 \vee \ldots L_n$, $D = K_1 \vee \ldots \vee K_m$ dos cláusulas. Definimos el *ST-árbol* $\mathcal{T}(C, D)$ mediante la definición recursiva del conjunto de nodos y de aristas del i-ésimo nivel, para $1 \leq i \leq n$. Los nodos serán parejas de naturales, además definimos una sustitución ϑ_{ij} correspondiente al nodo (i, j) . Para $i = 0$ sean $\mathcal{N}_0 = \{(0, 0)\}$, $\mathcal{A}_0 = \varnothing$ y $\vartheta_{00} = \varnothing$.
Sean $N \in \mathcal{N}_i, 1 \leq j \leq m$ y μ una sustitución sobre $V(L_{i+1}\vartheta_N)$ tal que $L_{i+1}\vartheta_N\mu = K_j$. Entonces $(i + 1, j) \in \mathcal{N}_{i+1}$ y $(N, (i + 1, j)) \in \mathcal{A}_i$, finalmente definimos $\vartheta_{(i+1)j} = \vartheta_N\mu$. Si no existe K_j tal que $L_{i+1}\vartheta_N \leq_s K_j$ entonces N es una hoja.

Ejemplo 4.4 Sean $C = Pxy \vee Pyz \vee Pxz$ y $D = Pab \vee Pbc \vee Pba$. Entonces $\vartheta_{00} = \varnothing$, $\vartheta_{11} = \{x/a, y/b\}$ y $L_1\vartheta_{11} = K_1$.
Existe $\mu = \{z/c\}$ tal que $L_2\vartheta_{11}\mu = K_2$, por lo tanto $(2, 2) \in \mathcal{N}_2$ y $\vartheta_{22} = \{x/a, y/b, z/c\}$. Después de calcular ϑ_{22} no existe K_j tal que $L_3\vartheta_{22} \leq_s K_j$, por lo tanto $(2, 2)$ es una hoja. En este momento se aplica el proceso de retroceso (*backtracking*) y ST produce $\vartheta_{23} = \{x/a, y/b, z/a\}$. Pero también se obtiene $L_3\vartheta_{23} \not\leq_s K_j$ para $j = 1, 2, 3$ y ST retrocede para definir ϑ_{12} y así sucesivamente.

Por definición de ST-árbol, $ST(C, D) = true$ syss existe un camino de longitud $|D|$ en $\mathcal{T}(C, D)$. Además el número de nodos en $\mathcal{T}(C, D)$ coincide con el número de sustituciones generadas por ST.

Para terminar la sección hacemos el análisis de correctud del algoritmo de Stillman.

Teorema 4.3 *ST es un algoritmo de decisión para subsunción, es decir, si $C \neq \square$ y D es cerrada entonces ST siempre termina y $ST(C, D) = true$ syss $C \leq_{ss} D$.*

dem:

Primero veamos que ST siempre termina. El número de llamadas anidadas de STI está acotado por $|C|$, además el máximo número de llamadas correspondientes a un nodo del ST-árbol es $|D|$. Por lo tanto el número total de llamadas a STI es menor o igual que $|NOD(\mathcal{T}(C,D)||D|$.
Para demostrar la correctud de ST hacemos inducción sobre $|C|$.

- $|C| = 1$. Calculamos $STI(C, D, 1, 1, \varnothing)$. El proceso entra en el ciclo-while con $a = 1$ y $\vartheta = \varnothing$. Tenemos dos casos:
  - Existe a tal que $a \leq |D|$ y $unif(L_1, K_a)$.
    En tal caso existe ϑ tal que $L_1\vartheta \in LIT(D)$ y como $C = L_1$ entonces $C \leq_{ss} D$. Pero como $|C| = 1$ entonces $ST(C, D) = STI(C, D, 1, 1, \varnothing) = true$.
  - Para toda $a \leq |D|$, $unif(L_1, K_a) = false$.
    El ciclo-while devuelve $a = |D| + 1$ y por lo tanto $ST(C, D) = STI(C, D, 1, 1, \varnothing) = false$. Como las K_a son cerradas y $unif(L_1, K_a)$ es falso entonces $L_1 \not\leq_s K_a$ para toda $K_a \in LIT(D)$, por lo tanto $L_1 = C \not\leq_{ss} D$.
- Hipótesis de Inducción. Si $|C| \leq k$ entonces $C \leq_{ss} D$ syss $ST(C, D) = true$.
- Sea C tal que $|C| = k + 1$ con $k \geq 1$. Tenemos dos casos.
  - La primera ejecución del ciclo-while devuelve $a = |D| + 1$.
    Por definición tenemos $STI(C, D, 1, 1, \varnothing) = ST(C, d) = false$, además como $a = |D| + 1$ entonces no existe ϑ tal que $L_1\vartheta \in LIT(D)$. Pero esto significa que $L_1 \not\leq_{ss} D$ lo cual implica que $C \leq_{ss} D$.
  - La primera ejecución del ciclo-while devuelve $a \leq |D|$ y el umg $\mu_1 = umg(L_1, K_a)$. Como $|C| \neq 1$ entonces se llama a $STI(C, D, 2, 1, \mu_1)$; obsérvese que si $C' = C \setminus L_1$ entonces $STI(C'\mu_1, D, 1, 1, \varnothing)$ devuelve el mismo valor que $STI(C, D, 2, 1, \mu_1)$ y como $|C'| = k$ entonces por HI $STI(C'\mu_1, D, 1, 1, \varnothing) = true$ syss $C'\mu_1 \leq_{ss} D$. Ahora bien, si $STI(C, D, 2, 1, \mu_1) = true$ entonces $STI(C, D, 1, 1, \varnothing) = true = ST(C, D)$, por definición de ST. Por otra parte tenemos $L_1\mu_1 \in LIT(D)$ y $LIT(C'\mu_1)\vartheta \subseteq LIT(D)$ para alguna ϑ, pues $C'\mu_1 \leq_{ss} D$. Como $L_1\mu_1$ es cerrada entonces $L_1\mu_1 = L_1\mu_1\vartheta$, por lo que $L_1\mu_1\vartheta \in LIT(D)$. De todo esto obtenemos $LIT(C)\mu_1\vartheta = LIT(L_1\mu_1\vartheta \vee C'\mu_1\vartheta) \subseteq LIT(D)$, es decir, $C \leq_{ss} D$.
    Si $STI(C, D, 2, 1, \mu_1) = false$ entonces podemos buscar una nueva sustitución μ_1. En general podemos obtener sustituciones μ_{1i} ejecutando el ciclo-while correspondiente a los nodos de nivel 1 de $\mathcal{T}(C, D)$. Si $STI(C, D, 2, 1, \mu_{i1}) = true$ para algún μ_{1i} se procede como arriba, obteniéndose $C \leq_{ss} D$ y por definición $ST(C, D) = true$. Así que supongamos que para cualquier μ_{1i} definido en el nivel 1 de $\mathcal{T}(C, D)$

se obtiene $STI(C, D, 2, 1, \mu_{i1}) = false$. Esto significa que para toda $K \in LIT(D)$, si $L_1\mu = K$ entonces $C'\mu \not\leq_{ss} D$. Ahora obsérvese lo siguiente

(⋆) $C \leq_{ss} D$ syss existe ϑ tal que $dom(\vartheta) = V(L_1)$, $L_1\vartheta \in LIT(D)$ y $(C \setminus L_1)\vartheta \leq_{ss} D$.

Pero STI produce todas las sustituciones ϑ tales que $dom(\vartheta) = V(L_1)$ y $L_1\vartheta \in LIT(D)$. Sea Λ el conjunto de tales sustituciones. En este caso Λ solo consiste de las μ_{1i}, así que $STI(C, D, 2, 1, \vartheta) = false$ para toda $\vartheta \in \Lambda$. Por lo tanto $ST(C'\vartheta, D, 1, 1, \varnothing) = false$ para toda $\vartheta \in \Lambda$, de donde por HI obtenemos que $C'\vartheta \not\leq_{ss} D$ para toda $\vartheta \in \Lambda$. Mediante (⋆) y la definición de Λ obtenemos que

$C \leq_{ss} D$ syss existe $\vartheta \in \Lambda$ tal que $C'\vartheta \leq_{ss} D$

Finalmente como no existe tal $\vartheta \in \Lambda$ concluimos que $C \not\leq_{ss} D$ y por definición de STI obtenemos $STI(C, D, 1, 1, \varnothing) = false$. Consecuentemente $ST(C, D) = false$ y la prueba está completa.

⊣

En este capítulo presentamos, mediante la técnica de subsunción, una manera de evitar redundancia y eliminar cláusulas innecesarias. Este proceso permite optimizar la búsqueda de una refutación, al reducir el número de cláusulas retenidas.
Además hemos mostrado que el proceso de subsunción es compatible con la regla de resolución, en el sentido de conservar la completud, lo cual justifica que la combinación de subsunción y resolución durante la búsqueda permanece correcta.
Con este capítulo terminamos la exposición de resultados que justifican los procesos implementados en OTTER, si bien alguna combinación de procesos podría resultar incompleta, como el uso de UR-resolución en cláusulas que no sean de Horn, su uso puede resultar de utilidad en casos particulares para resolver un problema específico, (véase [McC95], pag 43).

El siguiente y último capítulo se ocupa de describir, a grandes rasgos, la arquitectura de OTTER y de dar algunos ejemplos específicos que muestran los procesos descritos en los capítulos anteriores.

Capítulo 5

OTTER

En este capítulo presentaremos brevemente al programa de razonamiento automático OTTER (Organized Techniques for Theorem Proving and Effective Research), del laboratorio nacional de Argonne, Ill. USA. No pretendemos dar aquí un curso tutorial para su manejo; sólo haremos una descripción general de sus principales aspectos y daremos algunos ejemplos que utilizan las reglas de inferencia y procesos descritos anteriormente.
Para conocer a fondo todas las funciones del programa véase [McC95], además en [WOLB92] se puede encontrar un tutorial básico del programa, mientras que [Wos96] contiene un tutorial avanzado.
Antes de pasar a la descripción de OTTER hacemos un paréntesis histórico del desarrollo de programas de razonamiento automático en el laboratorio nacional de Argonne.

1963 Primera Generación. PG1 (Program 1), diseñado por Larry Wos, Dan Carson y George Robinson. Implementación de la estrategia de preferencia de unidades.[1]

1967 RG1, diseñado por Wos y George Robinson. Implementación de resolución binaria, factorización, paramodulación así como de la estrategia del conjunto de soporte.

1970 Segunda Generación iniciada por Ross Overbeek, estaba basada en NIUTP (Northern Illinois University Theorem Prover) programa que evolucionó en AURA (AUtomated Reasoning Assistant) y sus variantes. Contribuciones de Brian Smith, Rusty Lusk, Bob Veroff, Steve Winker y Wos. La generación NIUTP/AURA fue escrita principalmente en IBM ensamblador y PL/1 e incluía las primeras implementaciones eficientes de hiperresolución, demodulación, paramodulación e introdujo UR-resolución y peso. El resultado fue el primer conjunto de programas que produjo respuestas a

[1] Esta estrategia causa que un programa de razonamiento automático prefiera escoger al menos una cláusula unitaria como padre de una regla de inferencia.

diversos problemas abiertos en álgebra booleana y semigrupos entre otros temas.

1980 Tercera Generación. Iniciada por Overbeek, Lusk y William McCune. Consistía principalmente de LMA (Logic Machine Architecture) e ITP (Interactive Theorem Prover). LMA era una herramienta para construir sistemas de deducción, escrito en Pascal; ITP fue construido con LMA y su funcionalidad era similar a la de AURA. Las motivaciones para construir los dos programas fueron una ingeniería de software correcta y su portabilidad. Diversos demostradores experimentales basados en tecnología para compilar programas lógicos fueron también parte de la tercera generación.

1987 Cuarta Generación. En junio de 1987 McCune inicio la escritura de un nuevo demostrador; para diciembre ya habia suficiente codigo en lenguaje C útil para la investigación acerca de reglas de inferencia y estrategias de búsqueda. Tal programa fue llamado posteriormente OTTER. Otros miembros de la cuarta generación son ROO (Radical OTTER Optimization) que es una versión paralela de OTTER escrita por John Slaney, Lusk y McCune; FDB (Formula DataBase) una herramienta para unificación e indexación escrita por Overbeek y Ralph Butler; MACE (Models And CounterExamples) un buscador de modelos finitos escrito por McCune; y el más reciente programa de razonamiento automático de Argonne escrito también por McCune, EQP (Equational Theorem Prover) que es un programa para la lógica ecuacional que implementa AC-Unificación.

Para un poco más de historia véase [OL97].

5.1 Estrategia

Si bien resulta claro que los métodos de razonamiento automático basados en resolución son mucho mejores que aquellos basados en el teorema de Herbrand, como el presentado en [AM98], necesitamos mucho más para lograr que un programa de razonamiento automático sea efectivo; una de las cosas que esto requiere es el uso de *estrategias*. Una estrategia es una serie de metarreglas que guían la aplicación de las reglas de inferencia. Si una estrategia decide que regla de inferencia debe aplicarse en cierto momento del proceso, decimos que es una *estrategia de dirección*. Si una estrategia evita el uso de ciertas reglas de inferencia entonces decimos que es una *estrategia de restricción*. Ambos tipos de estrategia son necesarios para la efectividad de un programa de razonamiento automático.
Actualmente la estrategia de restricción más poderosa es la *estrategia del conjunto de soporte* [WRC65]. Intuitivamente dicha estrategia le prohibe a un programa de razonamiento automático aplicar una regla de inferencia a menos que uno de los padres o premisas se haya deducido o sea miembro de un conjunto distinguido de cláusulas de entrada. En la sección 2.7.2 hemos probado que esta estrategia resulta completa en combinación con resolución, por lo que tambien

lo será para hiperresolución. La completud de esta estrategia con paramodulación puede verse en [RW69].
En la práctica el conjunto de soporte debe escogerse con criterios semánticos para explotar con mayor efectividad la estrategia. Por ejemplo, si se desea probar un teorema de la forma $P_1 \wedge \ldots \wedge P_n \rightarrow Q$, basta probar la no satisfacibilidad de $\mathbb{S} = \{Cl(P_1), \ldots, Cl(P_n), Cl(\neg Q)\}$. Ahora bien, podemos identificar tres subconjuntos en $\mathbb{S}$, el conjunto de axiomas o hipótesis del área de estudio del teorema a probar; un conjunto de cláusulas particulares del teorema en particular (que ocasionalmente es vacío) y el conjunto de cláusulas que corresponde a la negación de la conclusión. Una elección adecuada para el conjunto de soporte es este último conjunto.

En contraste con las estrategias de restricción, una estrategia de dirección dicta hacia donde dirigir la atención. Una de las más poderosas estrategias de este tipo es la *estrategia del peso*. Para utilizar esta estrategia el usuario asigna prioridades o *pesos* a las variables, símbolos de función y de predicado y a los términos. De esta manera el usuario informa al programa acerca de qué conceptos considera claves para la resolución del problema en cuestión. A estas prioridades se les confía la dirección del programa en la elección de la siguiente cláusula a procesar. Un programa de razonamiento automático que emplea pesos elige la cláusula con el peso más favorable; en el caso de OTTER, se prefieren cláusulas más ligeras, es decir, cláusulas con menor peso.
La estrategia del peso puede utilizarse independientemente de las reglas de inferencia en acción, imponiendo una estrategia de dirección en el procesamiento de cláusulas. Además, el peso también pone un límite que causa la eliminación de claúsulas cuyo peso es mayor que el límite, de esta forma el peso impone también una estrategia de restricción.
En cualquier forma que sea usada, esta estrategia puede causar la pérdida de la completud refutacional. Como estrategia de restricción la completud refutacional puede perderse mediante la posibilidad de eliminar una cláusula demasiado pesada. Como estrategia de dirección la pérdida resultaría de la posibilidad de que el programa mantenga nuevas cláusulas tan ligeras que la cláusula necesaria nunca se examine. Sin embargo, si no hay límite en el peso y el programa analiza exhaustivamente todas las cláusulas, la completud refutacional no se perderá mediante el peso.
OTTER también ofrece una técnica efectiva para búsqueda de pruebas cortas conocida como la *estrategia de la razón*. Esta estrategia fue formulada para combinar la búsqueda en la base de datos de cláusulas retenidas, es decir, para escoger que cláusula debe ser la siguiente en procesar. Aunque el peso es de gran utilidad para guiar la búsqueda de un programa, su uso retrasa y hasta evita el procesamiento de cláusulas muy pesadas. Una manera para seleccionar con anticipación clásulas pesadas es utilizar la búsqueda a lo ancho (breadth-first search). Esta manera será exitosa al menos para las cláusulas de bajo nivel.[2]

[2]En referencia al nivel del arból de búsqueda, las cláusulas de entrada tienen nivel cero y una cláusula obtenida mediante inferencia tiene como nivel el sucesor del máximo de los

Sin embargo una búsqueda a lo ancho fuerza al programa a escoger en cada nivel un número mayor de cláusulas. Para combinar las dos estrategias tenemos la estrategia de la razón, que dado un entero k causa que el programa elija para procesar k cláusulas por peso y después una por nivel.
El estudio de nuevas y mejores estrategias es de gran importancia y es uno de los principales temas de investigación actualmente.

5.2 Arquitectura

OTTER está escrito en lenguaje C el cual proporciona velocidad y portabilidad. Consta de aproximadamente 35000 lineas de código (incluyendo comentarios). Las cláusulas y términos se almacenan en estructuras de datos compartidas, lo cual optimiza la indexación y las operaciones de inferencia y ahorra memoria. Para accesar términos y cláusulas para operaciones de subsunción, aplicaciones de reglas de inferencia y demodulación existen algoritmos de indexación especialmente diseñados. OTTER esta diseñado para correr en un ambiente UNIX aunque existen versiones limitadas que corren bajo DOS o Macintosh.

OTTER lee un archivo de entrada que contiene un conjunto de fórmulas o cláusulas de la lógica de primer orden con igualdad, dividido en listas y alguna información extra para controlar la ejecución. El conjunto de fórmulas incluye a las hipótesis y la negación de la conclusión (todas las pruebas son por contradicción); la información de control consiste de diversos parámetros y banderas que especifican las reglas de inferencia que se utilizarán y las estrategias de búsqueda. Mientras OTTER busca, va escribiendo la información obtenida a un archivo de salida, incluyendo la refutación buscada si esta se encontró. Las fórmulas se construyen con los siguientes símbolos:

negación	-
conjunción	&
disyunción	|
implicación	->
equivalencia	<->
cuantificación existencial	exists
cuantificación universal	all
igualdad	=
desigualdad	!=

Las reglas de inferencia que OTTER conoce son resolución binaria, hiperresolución, hiperresolución negativa y UR-resolución. El razonamiento ecuacional se hace mediante paramodulación y diversas formas de demodulación. La factorización, a diferencia de la manera en que la tratamos en el capitulo 2, no

niveles de sus padres.

está incorporada en la resolución sino que se considera como una regla independiente. Mediante el comando `set(ri)` se le indica al programa que regla de inferencia usar y mediante el comando `clear(ri)` se le indica que regla dejar de usar. Aquí `ri` puede ser una de las siguientes:

`binary_res`. Si esta bandera se habilita se utiliza la regla de resolución binaria para generar nuevas cláusulas.

`hyper_res`. Si esta bandera se habilita se utilizará la hiperresolución para generar nuevas cláusulas.

`neg_hyper_res`. Si se habilita se utilizará la hiperresolución negativa para generar nuevas cláusulas.

`ur_res`. Si se habilita, la regla de UR-resolución se utilizará para generar nuevas cláusulas.

`para_into`. Si se habilita se utilizará paramodulación hacia la *cláusula dada*[3] para generar nuevas cláusulas. Al usar paramodulación se debe incluir en la entrada la cláusula $x = x$.

`para_from`. Si se habilita se utilizará paramodulación desde la *cláusula dada* para generar nuevas cláusulas.

`demod_inf` Si se habilita se utilizará la demodulación como inferencia. Esto es útil cuando el objetivo principal es reescritura de términos. Si la bandera esta habilitada la cláusula dada se copia y procesa como cualquier cláusula recién generada.

En lo que respecta al control, un proceso de OTTER puede ser visto como una cerradura computacional con control de redundancia, es decir, OTTER intenta calcular la cerradura de un conjunto de enunciados de entrada bajo un conjunto de reglas de inferencia, aplicando reglas de eliminación y simplificación (subsunción y demodulación) que preservan la completud refutacional del sistema de inferencia. Tal proceso de cerradura es controlado por un ciclo que veremos en la siguiente sección.

5.2.1 El proceso de inferencia de OTTER

OTTER trabaja con cláusulas únicamente, pero es posible proporcionarle fórmulas cualesquiera como entrada, las cuales serán transformadas automáticamente a forma clausular. El mecanismo básico de inferencia de OTTER es el *algoritmo de la cláusula dada* que se puede ver como una implementación de la estrategia del conjunto de soporte. OTTER mantiene cuatro listas de cláusulas:

usable. En esta lista están las cláusulas disponibles para hacer inferencias.

[3] Cf. siguiente sección

sos. Es la lista del conjunto de soporte (set of support), las cláusulas de esta lista no estan disponibles para hacer inferencias por sí solas; están esperando para participar en la búsqueda de una refutación.

passive. Las cláusulas de esta lista no participan directamente en la búsqueda; se usan únicamente para subsunción y conflicto de unidades. Esta lista está fija desde el inicio y no cambia durante el proceso.

demodulators. Aquí están los demoduladores que se usan para reescribir las cláusulas inferidas.

Alguna de estas listas puede estar vacía. El ciclo de inferencia principal es el algoritmo de la cláusula dada que opera en las listas sos y usable

while sos $\neq \varnothing$ o no haya una refutación **do**
  Sea Cd la cláusula más ligera de sos;
  sos := sos $-$ {Cd};
  usable := usable $\cup$ {Cd};
  Infiere(Cd);
end while

Donde el proceso Infiere(Cd) genera nuevas cláusulas utilizando las reglas de inferencia en acción con la condición de que cada cláusula nueva debe tener a Cd como uno de sus padres y elementos de la lista usable como sus padres restantes. Además las nuevas cláusulas se someten a pruebas de retención y aquellas que sobreviven se agregan a la lista sos.

El proceso por el que pasa una cláusula recién inferida Cn es el siguiente:

1: Renombrar las variables de Cn;
2: * Imprimir Cn;
3: Aplicar demodulación a Cn;
4: * Orientar igualdades;
5: * Aplicar eliminación de cláusulas unitarias;
6: Fusionar literales idénticas (se mantiene la que está más a la izquierda);
7: * Simplificar mediante factorización;
8: * Eliminar Cn y terminar si Cn tiene demasiadas literales o variables;
9: Eliminar Cn y terminar si Cn es una tautología;
10: Eliminar Cn y terminar si Cn es muy pesada;
11: * Ordenar literales;
12: * Eliminar Cn y terminar si Cn es subsumida por alguna cláusula de usable $\cup$ sos $\cup$ passive (Subsunción hacia adelante);
13: Agregar Cn a sos;
14: * Imprimir Cn;
15: Si Cn tiene cero literales entonces se halló una refutación;

16: Si Cn tiene una literal, buscar en usableUsosUpassive un conflicto de unidades (refutación) con Cn;
17: * Imprimir la demostración si se halló una refutación;
18: * Intentar agregar Cn como demodulador;
19: ◇ Si se aplicó el paso 18 demodular hacia atrás;
20: ◇ Eliminar cada cláusula de usable ∪ sos que sea subsumida por Cn (Subsunción hacia atrás);
21: ◇ Factorizar Cn y procesar los factores.

Los pasos marcados con * son opcionales y los marcados con ◇ son pasos opcionales que se aplican hasta que los anteriores se aplicaron a todas las cláusulas obtenidas mediante Infiere(Cd).
La eliminación de cláusulas unitarias a las que se refiere el paso 5 consiste en eliminar una literal de una cláusula recién generada si es la negación de una instancia de una cláusula unitaria que figura en usable ∪ sos. Por ejemplo, la segunda literal de $P(a,x) \vee Q(a,x)$ es eliminada por la literal $\neg Q(u,v)$, pero no se elimina mediante $\neg Q(u,b)$ pues la unificación requiere la instanciación de x.
El paso 11 ordena las literales de manera que las negativas aparezcan antes que las positivas.

5.2.2 El Modo Autónomo

Aunque OTTER no es un programa interactivo, se utiliza casi siempre de esta manera. Realizamos una búsqueda, examinamos la salida, cambiamos las banderas para ajustar el comportamiento e iniciamos una nueva búsqueda. Si bien el proceso de elegir nuevos parámetros para reiniciar una búsqueda no puede ser automatizado aún, OTTER ofrece un modo de hacerlo bastante efectivo en algunos casos, llamado el *modo autónomo* que es útil para usuarios inexpertos así como para resolver problemas sencillos o para situaciones en las que OTTER es llamado desde otro programa.
En el modo autónomo, OTTER habilita incondicionalmente diversas opciones y particiona las cláusulas de entrada en las listas usable y sos. Después analiza la entrada para verificar si todas las cláusulas son proposicionales, si son de Horn, si la igualdad esta presente y en tal caso si hay axiomas de igualdad. El algoritmo del modo autónomo es el siguiente:

```
max_mem := 12 megabytes;
set(control_memory);
pick_given_ratio := 4;
set(process_input);
Poner las cláusulas positivas en sos;
Poner las cláusulas negativas en usable;
if todas las cláusulas son proposicionales then
  set(propositional);
  set(hyper_res);
else
  if hay cláusulas no unitarias then
```

```
    set(hyper_res);
    if todas las cláusulas son de Horn then
      clear(ordered_hyper).
    else
      set(factor);
      set(unit_deletion);
    end if
  end if
  if hay igualdades then
    set(knuth_bendix);
    if hay axiomas de igualdad then
      clear(paramodulation_flags);
    end if
  end if
end if
```

El parámetro `max_mem` limita la memoria disponible para almacenamiento de cláusulas y estructuras de datos relacionadas, el parámetro `pick_given_ratio` implementa la estrategia de la razón, la bandera `process_input` causa que las cláusulas de entrada se procesen como cláusulas derivadas, `propositional` provoca que se efectuen diversas optimizaciones, la bandera `ordered_hyper` habilitada por default evita que los satélites se resuelvan con literales no maximales, la bandera `factor` habilita la factorización, la bandera `unit_deletion` provoca que cada cláusula unitaria digamos P se utilice como regla de reescritura $P = TRUE$ para simplificar cláusulas no unitarias derivadas, la bandera `knuth_bendix` provoca que se habiliten diversas opciones para que la búsqueda se comporte como un proceso de compleción de Knuth-Bendix, con la frase `paramodulation_flags` indicamos diversas banderas de paramodulación. Para una mejor descripción de estos conceptos véase [McC95].

5.3 Literales de Respuesta

Muchas veces se le puede presentar a un programa de razonamiento automático una pregunta existencial. En este caso tal vez no es suficiente con hallar una refutación, sino que nos gustaría hallar la respuesta a tal pregunta. Por ejemplo, si tenemos las cláusulas $P(a,b), P(b,c), \neg P(x,y) \vee \neg P(y,z) \vee A(x,z)$ donde P significa "ser padre de" y A "ser abuelo de" y nos preguntamos si $A(x,z)$ entonces nos gustaría saber no solamente si existe un par en la relación "ser abuelo de" sino también saber cual es. Esto es lo que hacen las literales de respuesta. La cláusula $\neg A(x,z)$ representa a la negación de la pregunta $\exists x, z\ A(x,z)$, pero si se le proporciona al programa esta cláusula junto con las anteriores, este únicamente respondera con la cláusula vacía sin producir la respuesta explícita de quienes son x y z. Si agregamos a la pregunta una literal de respuesta denotada $Ans(x,z)$ entonces al ejecutar el programa este no terminará con $\square$ sino con $Ans(a,c)$ como era de esperarse.

En general si la pregunta es $\exists x P(x)$ entonces la cláusula correspondiente es

$\neg P(x) \vee Ans(x)$ la cual corresponde a la fórmula $\forall x(P(x) \rightarrow Ans(x))$ que significa que *si el objeto x cumple P entonces "x" es la respuesta que necesitamos.* Si se halla una refutación entonces en lugar de la cláusula vacía obtendremos en la literal de respuesta el objeto cuya existencia se ha probado.
El uso principal de literales de respuesta es para recordar instancias de variables realizadas durante la busqueda por una refutación. En OTTER una literal de respuesta se denota con los símbolos $ans,$Ans o $ANS. La mayoría de las rutinas de OTTER incluyendo las que cuentan cláusulas o deciden si una cláusula es positiva o negativa ignoran las literales de respuesta, además OTTER nunca intenta factorizar literales de respuesta.
En matemáticas es práctica común encontrar la solución de un problema simplemente probando su existencia. Por ejemplo, para encontrar la solución de una ecuación primero se puede probar que ésta existe y tal vez a partir de esta prueba hallemos el objeto que representa la solución. Veamos un ejemplo al respecto. En lenguaje de OTTER escribiremos, los comentarios inician con un signo %.

Ejemplo 5.1 Tenemos los siguientes axiomas.

$\forall x \forall y \forall z\ ((x + y) + z = x + (y + z))$ Asociatividad.

$\forall x \forall y \exists z\ (z + x = y)$ Existencia de solución por la izquierda.

$\forall x \forall y \exists z\ (x + z = y)$ Existencia de solución por la derecha.

y queremos preguntarnos si hay una identidad derecha, es decir, si

$$\exists x \forall y\ (y + x = y)$$

Proporcionamos a OTTER el siguiente programa de entrada.

```
% Iniciamos el modo autónomo en OTTER 3.0.5, si se usa OTTER 3.0.4 hay que
%poner set(auto).

    set(auto2).

% Se colocan los axiomas en la lista usable como fórmulas,utilizando el co-
mando list(usable), si se desean dar las % cláusulas directamente se debe
usar list en lugar de formula_list. La lista debe terminarse con el comanto
end_of_list.

    formula_list(usable).

    all x y z (f(f(x,y),z)=f(x,f(y,z))).

    all x y exists z (f(z,x)=y).

    all x y exists z (f(x,z)=y).

    end_of_list.
```

% Se coloca la negación de la pregunta ya en forma clausular junto con la literal
% de respuesta en la lista sos. Obsérvese que h es una función de Skolem.

```
list(sos).

f(h(x),x) != h(x) | $Ans(x).

end_of_list.
```

% Hasta aquí el programa de entrada.

Despues de unos segundos OTTER devuelve un archivo de salida el cual contiene entre otras cosas la lista usable transformada a cláusulas de la siguiente manera:

```
list(usable).

f(f(x,y),z)=f(x,f(y,z)).

f($f1(x,y),x)=y.

f(x,$f2(x,y))=y.

end_of_list.
```

donde las funciones $f1 y $f2 son funciones de Skolem. Además en lugar de la cláusula vacía la respuesta es $Ans($f2(y,y)). Lo cual significa que la identidad buscada es $f2(y,y).
La prueba que también aparece en el programa de salida se analizará en la siguiente sección.

Para una visión más general del uso de las literales de respuesta así como para ver más ejemplos sugerimos consultar el capítulo 11 de [CL73]. Para cuestiones de la semántica de las literales de respuesta el artículo [Kun96] es muy accesible después de haber leido esta sección y el primer capítulo de este trabajo.

5.4 Ejemplos

Para culminar este trabajo presentamos algunos ejemplos sencillos que muestran las principales funciones de OTTER. Todos los ejemplos son tomados de [CL73, Lov78, McC95] pero la implementación y elección de reglas en cada caso es aportación nuestra.

OTTER manda su salida a la salida estandar pero es más comodo redireccionar la salida hacia un archivo, por ejemplo si el programa de entrada se llama `pro1.in`, mediante la instrucción

```
[ekaterina]$ [4] otter < pro1.in > pro1.out
```

[4] Esto denota al prompt de UNIX

se genera un archivo `pro1.out` que contiene los resultados obtenidos por el programa.
La primera parte del archivo de salida contiene a la entrada así como alguna información adicional, como las cláusulas obtenidas, si se dieron fórmulas como entrada, identificación de cláusulas mediante una numeración y las banderas habilitadas si se usó el modo autónomo. La segunda parte del archivo de salida refleja la búsqueda, en esta parte figuran las **cláusulas dadas**, **cláusulas retenidas** así como diversos mensajes acerca del procesamiento.
La tercera parte del archivo consta de la o las pruebas halladas mientras que la última parte presenta diversas estadísticas.
Para entender una prueba de OTTER debemos explicar que significa la lista que precede a cada cláusula. Cada cláusula C que figura en la salida tiene la siguiente forma:

$$n\ [\ldots]\ C.$$

donde n es un número llamado el identificador de la cláusula y la lista $[\ldots]$ consta de lo siguiente. Si la lista es vacía entonces se trata de una cláusula de entrada, si no el primer elemento de la lista puede ser:

- Una regla de inferencia. Indicando que la cláusula se generó mediante tal regla de inferencia, los identificadores de los padres se listan después del nombre de la regla de inferencia, listando en primer lugar a la cláusula dada.

- Un identificador de cláusula. Indicando que la cláusula fue generada por demodulación, siempre y cuando la bandera `demod_inf`, que indica el uso de demodulación como regla de inferencia, esté habilitada.

- `new_demod`. Indicando que la cláusula es un demodulador recién generado; esta cláusula es una copia de la cláusula cuyo identificador se lista despues de `new_demod`.

- `back_demod`. Indicando que la cláusula se generó demodulando hacia atrás la cláusula cuyo identificador figura inmediatamente después de `back_demod`.

- `demod`. Indicando que la cláusula se generó demodulando hacia atrás una cláusula de entrada.

- `factor_simp` Indicando que la cláusula fue generada factorizando una cláusula de entrada.

la sublista [demod,$Id_1,\ldots,Id_n$] indica demodulación con $Id_1,\ldots,Id_n$, la sublista [unit_del,$Id_1,\ldots,Id_n$] indica eliminación de unidades con $Id_1,\ldots,Id_n$.
Si la bandera `detailed_history` está habilitada entonces al utilizar las reglas `para_from`, `para_into`, `binary_res` las posiciones de las literales o términos unificados se listan junto con los identificadores de los padres. Por ejemplo,

[binary, 57.3, 30.2] significa que la tercera literal de la cláusula 57 se resolvió con la segunda literal de la cláusula 30.
Para la paramodulación la cláusula "desde" se lista como $Id.i.j$ donde i es el número de la literal de igualdad y j es 1 o 2, que indica el número del término que se esta unificando en la igualdad; la cláusula "hacia" se lista como $Id.i.j_1.....j_n$ donde i es el número de la literal del término "hacia" y $j_1.....j_n$ es el vector de posición del término "hacia"; por ejemplo, 400.3.1.2 denota al segundo argumento del primer argumento de la tercera literal de la cláusula 400. Veamos un ejemplo trivial para analizar el archivo de salida y entender la notación.

Ejemplo 5.2 Considérese el siguiente programa de entrada:

```
set(binary_res).
set(para_into).

list(usable).
P(x)| -Q(x,y).
Q(a,b).
a=f(b,c).
end_of_list.

list(sos).
-P(f(b,c)).
end_of_list.
```

OTTER produce inmediatamente el siguiente archivo de salida, los comentarios iniciados con % no figuran en el archivo, los agregamos para explicar los componentes del archivo

```
----- Otter 3.0.5, Feb 1998 -----
The process was started by favio on ekaterina,
Wed Sep 16 15:45:46 1998
The command was "otter".  The process ID is 688.

set(binary_res).
   dependent: set(factor).
   dependent: set(unit_deletion).
set(para_into).

list(usable).
1 [] P(x)| -Q(x,y).
2 [] Q(a,b).
3 [] a=f(b,c).
end_of_list.

list(sos).
```

```
4 [] -P(f(b,c)).
end_of_list.

======= end of input processing =======
```

% Hasta aquí la primera parte del archivo de salida, el cual incluye simplemente a la entrada y habilita las banderas de factorización y eliminación de unidades por ser dependientes de `binary_res`.

% A continuación mostramos la segunda parte del archivo, que incluye a la generación de cláusulas mediante el algoritmo de la cláusula dada (given clause) y a la refutación.

```
========== start of search ==========

given clause #1: (wt=4) 4 [] -P(f(b,c)).
** KEPT (pick-wt=5): 5 [binary,4.1,1.1] -Q(f(b,c),x).
** KEPT (pick-wt=2): 6 [para_into,4.1.1,3.1.2] -P(a).

given clause #2: (wt=2) 6 [para_into,4.1.1,3.1.2] -P(a).
** KEPT (pick-wt=3): 7 [binary,6.1,1.1] -Q(a,x).

----> UNIT CONFLICT at   0.03 sec ----> 8 [binary,7.1,2.1] $F.

Length of proof is 2.  Level of proof is 2.
```

% con `pick-wt` o `wt` se denota el peso de cada cláusula. `UNIT CONFLICT` quiere decir que se generó la cláusula vacía denotada con `$F`. En este caso hubo dos cláusulas dadas (#1 y #2) y la cláusula vacía se hallo en 0.03 segundos.

% A continuación viene la prueba, con la notación ya explicada.

```
---------------- PROOF ----------------

1 [] P(x)| -Q(x,y).
2 [] Q(a,b).
3 [] a=f(b,c).
4 [] -P(f(b,c)).
6 [para_into,4.1.1,3.1.2] -P(a).
7 [binary,6.1,1.1] -Q(a,x).
8 [binary,7.1,2.1] $F.

------------ end of proof -------------
```

```
Search stopped by max_proofs option.

============ end of search ============
```

% Hasta aquí la segunda parte del archivo de salida. La búsqueda terminó pues el parámetro max_proofs vale por default uno, lo cual indica que se pare la búsqueda en cuanto se halle la primera refutación.
% La tercera parte del archivo de salida mostrada a continuación simplemente proporciona diversas estadisticas.

```
-------------- statistics -------------
clauses given                  2
clauses generated              3
  binary_res generated         2
  para_into generated          1
  factors generated            0
demod & eval rewrites          0
clauses wt,lit,sk delete       0
tautologies deleted            0
clauses forward subsumed       0
  (subsumed by sos)            0
unit deletions                 0
factor simplifications         0
clauses kept                   3
new demodulators               0
empty clauses                  1
clauses back demodulated       0
clauses back subsumed          0
usable size                    5
sos size                       2
demodulators size              0
passive size                   0
hot size                       0
Kbytes malloced               63

----------- times (seconds) -----------
user CPU time          0.04          (0 hr, 0 min, 0 sec)
system CPU time        0.00          (0 hr, 0 min, 0 sec)
wall-clock time        0             (0 hr, 0 min, 0 sec)
input time             0.01
  clausify time        0.00
binary_res time        0.00
para_into time         0.00
pre_process time       0.01
  renumber time        0.00
```

```
  demod time          0.00
  order equalities    0.00
  unit deleletion     0.00
  factor simplify     0.00
  weigh cl time       0.00
  hints keep time     0.00
  sort lits time      0.00
  forward subsume     0.00
  delete cl time      0.00
  keep cl time        0.00
    hints time        0.00
  print_cl time       0.00
  conflict time       0.01
  new demod time      0.00
post_process time     0.00
  back demod time     0.00
  back subsume        0.00
  factor time         0.00
  unindex time        0.00

That finishes the proof of the theorem.

Process 688 finished Wed Sep 16 15:45:46 1998
```

Por último presentamos algunos ejemplos más interesantes, nos limitamos a dar los programas de entrada completos y la prueba que proporciona el archivo de salida.

Ejemplo 5.3 Tenemos el siguiente problema de geometría tomado de [Lov78].
Problema: Un triángulo isosceles tiene dos ángulos iguales.
a, b, c representan los vértices del triángulo.
$T(x, y, z)$ significa que los puntos x, y, z forman un triángulo.
$C(u, v, w, x, y, z)$ significa que el triángulo uvw es congruente con el triángulo xyz.
$S(u, v, x, y)$ significa que el segmento uv es igual al segmento xy.
$A(u, v, w, x, y, z)$ significa que el ángulo uvw es igual al ángulo xyz.

El programa de entrada es el siguiente:

```
%set(auto2).      %Clausula vacia en 49.

%set(hyper_res). %Clausula vacia en 36

%set(binary_res).  %Clausula vacia en 64.

%set(neg_hyper_res). %Clausula vacia en 56.
```

```
set(ur_res).  %Clausula vacia en 32.

list(sos).
T(a,b,c).
S(a,c,b,c).
-A(c,a,b,c,b,a).
end_of_list.

list(usable).
S(x,y,y,x).
-S(u,v,x,y)| S(x,y,u,v).
-S(u,v,x,y)| S(v,u,x,y).
-T(x,y,z)| T(y,z,x).
-T(x,y,z)| T(y,x,z).
-C(u,v,w,x,y,z)| A(u,v,w,x,y,z).
-C(u,v,w,x,y,z)| A(v,u,w,y,x,z).
-C(u,v,w,x,y,z)| A(u,w,v,x,z,y).
-S(u,v,x,y)| -S(v,w,y,z)| -S(u,w,x,z)| -T(u,v,w)
| -T(x,y,z)| C(u,v,w,x,y,z).
end_of_list.
```

Dependiendo de la regla que se utilice para hacer inferencias el número de cláusulas generadas crece. A continuación presentamos la prueba con UR-resolución que es la que produjo menos cláusulas, resultando más eficiente que el modo autónomo que halló la refutación en 49 pasos.

```
---------------- PROOF ----------------

1 [] T(a,b,c).
2 [] S(a,c,b,c).
3 [] -A(c,a,b,c,b,a).
4 [] S(x,y,y,x).
5 [] -S(u,v,x,y)|S(x,y,u,v).
6 [] -S(u,v,x,y)|S(v,u,x,y).
7 [] -T(x,y,z)|T(y,z,x).
8 [] -T(x,y,z)|T(y,x,z).
11 [] -C(u,v,w,x,y,z)|A(u,w,v,x,z,y).
12 [] -S(u,v,x,y)| -S(v,w,y,z)| -S(u,w,x,z)| -T(u,v,w)| -T(x,y,z)
      |C(u,v,w,x,y,z).
14 [ur,1,7] T(b,c,a).
16 [ur,14,8] T(c,b,a).
17 [ur,14,7] T(c,a,b).
18 [ur,2,6] S(c,a,b,c).
20 [ur,18,5] S(b,c,c,a).
24 [ur,20,6] S(c,b,c,a).
```

```
27 [ur,24,5] S(c,a,c,b).
28 [ur,27,12,24,4,16,17] C(c,b,a,c,a,b).
31 [ur,3,11] -C(c,b,a,c,a,b).
32 [binary,31.1,28.1] $F.

------------ end of proof -------------
```

Ejemplo 5.4 Todo grupo de exponente dos es abeliano.
Con este ejemplo mostramos el uso de la paramodulación desde y hacia la cláusula dada.

```
%set(auto2).

set(para_from).
set(para_into).

set(process_input).

% Con la siguiente bandera se analizaran todas las igualdades
% generadas como posibles demoduladores.

set(dynamic_demod_all).

list(sos).   % En este caso se soportan todas las clausulas.
x=x.         % Agregamos el axioma de reflexividad.
f(e,x)=x.
f(x,g(x))=e.
f(f(x,y),z)=f(x,f(y,z)).
f(x,x)=e.
f(a,b) != f(b,a).
end_of_list.
```

A continuación presentamos la prueba obtenida con este programa de entrada.

```
---------------- PROOF ----------------

3,2 [] f(e,x)=x.
4 [] f(x,g(x))=e.
6 [] f(f(x,y),z)=f(x,f(y,z)).
8 [] f(x,x)=e.
10 [] f(a,b)!=f(b,a).
11 [copy,10,flip.1] f(b,a)!=f(a,b).
14 [para_into,6.1.1.1,4.1.1,demod,3,flip.1] f(x,f(g(x),y))=y.
```

```
16 [para_into,6.1.1.1,8.1.1,demod,3,flip.1] f(x,f(x,y))=y.
20 [para_into,6.1.1,8.1.1,flip.1] f(x,f(y,f(x,y)))=e.
25,24 [para_into,16.1.1.2,4.1.1] f(x,e)=g(x).
27,26 [para_into,16.1.1.2,8.1.1,demod,25] g(x)=x.
30 [para_from,20.1.1,14.1.1.2,demod,25,27,27,flip.1]
                                        f(x,f(y,x))=y.
34 [para_from,30.1.1,14.1.1.2,demod,27] f(x,y)=f(y,x).
35 [binary,34.1,11.1] $F.

------------ end of proof -------------
```

Si no se hubieran procesado primero las cláusulas de entrada, omitiendo la bandera process_input, la cláusula vacía se hallaría en 79 pasos y no en 35.

Ejemplo 5.5 Mostrar que el conjunto

$$\mathbb{S} = \{x = a \vee x = b, \neg(g(x) = x), R(a,b), R(g(a),g(b)), \neg R(a,g(a)) \vee \neg R(b,g(b))\}$$

es no satisfacible.
El programa de entrada es el siguiente.

```
set(auto2).

%set(para_into).
%set(binary_res).
%set(back_demod).

list(usable).
x=x.
x=a| x=b.
g(x)!= x.
R(a,b).
R(g(a),g(b)).
end_of_list.


list(sos).
-R(a,g(a))| -R(b,g(b)).
end_of_list.
```

Vamos a presentar tres pruebas, la primera usa resolución binaria y paramodulación hacia la cláusula dada. Se obtuvo la refutación en 1050 pasos y 8.72 segundos, se generaron 1962 cláusulas, 52 por resolución, 1208 por paramodulación 702 por factorización. Además se subsumieron 1006 cláusulas.

```
---------------- PROOF ----------------

2 [] x=a|x=b.
3 [] g(x)!=x.
4 [] R(a,b).
5 [] R(g(a),g(b)).
6 [] -R(a,g(a))| -R(b,g(b)).
9 [para_into,6.2.1,2.2.2] -R(a,g(a))| -R(x,g(b))|x=a.
35 [binary,9.2,5.1,unit_del,3] -R(a,g(a)).
36 [para_into,35.1.1,2.1.2] -R(x,g(a))|x=b.
39 [para_into,36.1.2.1,2.1.2] -R(x,g(y))|x=b|y=b.
40 [para_into,36.2.2,36.2.2] -R(x,g(a))|x=y| -R(y,g(a)).
41 [para_into,36.2.2,2.2.2] -R(x,g(a))|x=y|y=a.
42 [factor,39,2,3] -R(x,g(x))|x=b.
58 [para_into,42.2.2,2.2.2] -R(x,g(x))|x=y|y=a.
355 [binary,41.3,3.1] -R(x,g(a))|x=g(a).
423 [binary,58.3,3.1] -R(x,g(x))|x=g(a).
857 [para_into,40.1.2,355.2.2] -R(x,y)|x=z| -R(z,g(a))
    | -R(y,g(a)).
874 [factor,857,3,4] -R(x,y)|x=y| -R(y,g(a)).
882 [binary,874.1,4.1] a=b| -R(b,g(a)).
924 [para_into,882.1.1,2.1.2,factor_simp] x=b| -R(b,g(a)).
959 [binary,924.1,3.1] -R(b,g(a)).
976 [para_into,959.1.1,36.2.2,factor_simp] -R(x,g(a)).
998 [para_into,976.1.2.1,2.1.2] -R(x,g(y))|y=b.
999 [para_into,976.1.2,423.2.2] -R(x,y)| -R(y,g(y)).
1022 [binary,999.1,4.1] -R(b,g(b)).
1030 [para_into,1022.1.2.1,998.2.2,factor_simp] -R(b,g(x)).
1037 [para_into,1030.1.1,2.2.2] -R(x,g(y))|x=a.
1049 [binary,1037.1,5.1] g(a)=a.
1050 [binary,1049.1,3.1] $F.

------------ end of proof -------------
```

En la segunda prueba se consigue la refutación en 508 pasos y 1.89 segundos simplemente agregando la demodulación hacia atrás en la segunda prueba.

```
---------------- PROOF ----------------

2 [] x=a|x=b.
3 [] g(x)!=x.
4 [] R(a,b).
5 [] R(g(a),g(b)).
6 [] -R(a,g(a))| -R(b,g(b)).
```

```
9 [para_into,6.2.1,2.2.2] -R(a,g(a))| -R(x,g(b))|x=a.
35 [binary,9.2,5.1,unit_del,3] -R(a,g(a)).
36 [para_into,35.1.1,2.1.2] -R(x,g(a))|x=b.
37 [para_into,35.1.2.1,2.1.2] -R(a,g(x))|x=b.
39 [para_into,36.1.2.1,2.1.2] -R(x,g(y))|x=b|y=b.
41 [para_into,36.2.2,2.2.2] -R(x,g(a))|x=y|y=a.
49 [para_into,37.2.2,2.2.2] -R(a,g(x))|x=y|y=a.
353 [binary,41.3,3.1,flip.2] -R(x,g(a))|g(a)=x.
375 [para_into,353.1.2.1,2.1.2] -R(x,g(y))|g(a)=x|y=b.
381 [factor,375,2,3] -R(b,g(g(a)))|g(a)=b.
393 [para_into,381.1.1,39.2.2,factor_simp,factor_simp]
                                        -R(x,g(g(a)))|g(a)=b.
460 [binary,49.3,3.1,flip.2] -R(a,g(x))|g(a)=x.
488 [para_into,460.1.1,2.1.2] -R(x,g(y))|g(a)=y|x=b.
491 [para_into,460.1.2,393.2.1,unit_del,4,3] -R(x,g(g(a))).
500,499 [factor,488,2,3,unit_del,5] g(a)=b.
507 [back_demod,491,demod,500] -R(x,g(b)).
508 [binary,507.1,5.1] $F.

------------ end of proof -------------
```

La tercera prueba muestra la efectividad del modo autónomo mediante el cual se obtiene una prueba con hiperresolución y demodulación hacia atrás en sólo 12 pasos:

```
---------------- PROOF ----------------

1 [] g(x)!=x.
2 [] -R(a,g(a))| -R(b,g(b)).
4 [] x=a|x=b.
5 [] R(a,b).
6 [] R(g(a),g(b)).
8,7 [hyper,4,1] g(a)=b.
10,9 [hyper,4,1] g(b)=a.
11 [back_demod,6,demod,8,10] R(b,a).
12 [back_demod,2,demod,8,10,unit_del,5,11] $F.

------------ end of proof -------------
```

Los siguientes dos ejemplos muestran el uso de las literales de respuesta.

Ejemplo 5.6 En este ejemplo mostramos la prueba del ejemplo 5.1.

```
---------------- PROOF ----------------

1 [] f(h(x),x)!=h(x)|$Ans(x).
2 [] f(f(x,y),z)=f(x,f(y,z)).
```

```
4 [] f($f1(x,y),x)=y.
6 [] f(x,$f2(x,y))=y.
9 [para_into,2.1.1.1,4.1.1,flip.1] f($f1(x,y),f(x,z))=f(y,z).
18,17 [para_into,9.1.1.2,6.1.1] f($f1(x,y),z)=f(y,$f2(x,z)).
26 [back_demod,4,demod,18] f(x,$f2(y,y))=x.
28 [binary,26.1,1.1] $Ans($f2(y,y)).

------------ end of proof -------------
```

Ejemplo 5.7 Veamos la solución al famoso problema del chango y el plátano utilizando literales de respuesta.
$P(x,y,z,u)$ denota que en el estado u, el chango está en la posición x, el plátano en y y la silla en z.
$R(x)$ denota que en el estado x el chango puede alcanzar el plátano.
$camina(y,z,u)$ denota al estado obtenido si el chango estaba en el estado u y camina de y a z.
$carga(y,z,u)$ denota al estado obtenido si el chango estaba en el estado u y camina de y a z cargando la silla.
$sube(u)$ es el estado obtenido si el chango estaba en el estado u y sube a la silla.

Suponemos que el chango está en la posición a, el plátano en la posición b, la silla en la posición c y que el chango está en el estado $s1$. El programa de entrada es el siguiente:

```
set(auto2).

list(sos).
P(a,b,c,s1).
- R(u)| $ANS(u).
end_of_list.

list(usable).
-P(x,y,z,u)| P(z,y,z, camina(x,z,u)).
-P(x,y,x,u)| P(y,y,y, carga(x,y,u)).
-P(b,b,b,u)| R(sube(u)).
end_of_list.
```

La prueba obtenida mediante el modo autónomo es:

```
---------------- PROOF ----------------

1 [] -P(x,y,z,u)|P(z,y,z,camina(x,z,u)).
2 [] -P(x,y,x,z)|P(y,y,y,carga(x,y,z)).
3 [] -P(b,b,b,x)|R(sube(x)).
4 [] P(a,b,c,s1).
5 [] -R(x)|$ANS(x).
6 [hyper,4,1] P(c,b,c,camina(a,c,s1)).
```

```
7 [hyper,6,2] P(b,b,b,carga(c,b,camina(a,c,s1))).
9 [hyper,7,3] R(sube(carga(c,b,camina(a,c,s1)))).
10 [binary,9.1,5.1] $ANS(sube(carga(c,b,camina(a,c,s1)))).

------------ end of proof -------------
```

De manera que si el chango camina desde a hasta c toma la silla y camina hasta b cargando la silla y sube a la silla podrá alimentarse.

Para finalizar el capítulo veamos como OTTER resuelve el problema de completación de Knuth-Bendix de la sección 3.4.

Debe quedar claro que mediante la bandera `knuth_bendix` OTTER simplemente altera una serie de opciones para que la búsqueda se comporte como un procedimiento de completación. OTTER no fue diseñado como un sistema de completación de manera que tal implementación no ha sido optimizada. Sin embargo, si al colocar el conjunto de ecuaciones a completar junto con $x = x$ en la lista sos, la búsqueda termina con la lista sos vacía, entonces la lista de demoduladores contiene la completación del conjunto original.

Ejemplo 5.8 Queremos completar el conjunto

$$E = \{e * x = x, i(x) * x = e, (x * y) * z = x * y * z\}$$

El programa de entrada es el siguiente:

```
set(knuth_bendix).

set(print_lists_at_end).

list(sos).
x=x.
e*x=x.
i(x)*x=e.
(x*y)*z=x*y*z.
end_of_list.
```

En este caso OTTER termina con el mensaje "sos empty" y las listas finales son:

```
Search stopped because sos empty.

========== end of search ==========

list(usable).
1 [] x=x.
```

```
2 [] e*x=x.
4 [] i(x)*x=e.
6 [] (x*y)*z=x*y*z.
8 [para_into,6.1.1.1,4.1.1,demod,3,flip.1] i(x)*x*y=y.
14 [para_into,8.1.1.2,4.1.1,demod,11] x*e=x.
18 [para_into,14.1.1,4.1.1,flip.1] i(e)=e.
20 [para_into,10.1.1,14.1.1,demod,15] i(i(x))=x.
24 [para_into,10.1.1,4.1.1,flip.1] x*i(x)=e.
22 [para_into,10.1.1,8.1.1,flip.1] x*i(x)*y=y.
42 [para_from,34.1.1,8.1.1.2,flip.1] i(x*y)=i(y)*i(x).
end_of_list.

list(sos).
end_of_list.

list(demodulators).
3 [new_demod,2] e*x=x.
5 [new_demod,4] i(x)*x=e.
7 [new_demod,6] (x*y)*z=x*y*z.
9 [new_demod,8] i(x)*x*y=y.
15 [new_demod,14] x*e=x.
19 [new_demod,18] i(e)=e.
21 [new_demod,20] i(i(x))=x.
23 [new_demod,22] x*i(x)*y=y.
25 [new_demod,24] x*i(x)=e.
43 [new_demod,42] i(x*y)=i(y)*i(x).
end_of_list.
```

Con estos ejemplos hemos mostrado los principales mecanismos de OTTER. Otros ejemplos sencillos pueden verse en [AM98]. Para ver la gran variedad de aplicaciones, tanto a las matemáticas como a las ciencias de la computacion, consúltese [WOLB92] o la página web de OTTER. Además una gran variedad de teoremas importantes de la teoría de conjuntos de Von-Neumann-Bernays-Gödel, la aritmética de Peano, la geometría de Tarski y los teoremas de incompletud de Gödel con ayuda de la lógica modal $K4$ pueden consultarse en la tesis doctoral de Art Quaife, reproducida en [Qua92].
Queremos terminar este trabajo con una tesis bastante atrevida del mismo Quaife:

Cualquier problema formulable matemáticamente y que puede ser resuelto, podrá ser resuelto por un programa de razonamiento automático.

¿ Cierto, falso, inverosímil, concebible ? la pregunta queda en el aire.

Bibliografía

[AGM92] S. Abramsky, D. Gabbay, T.S.E. Maibaum (Editores). Handbook of Logic in Computer Science, Vol. 2. Background: Computational Structures. Oxford Science Publications. 1992.

[AN89] H. Aït-Kaci, M. Nivat, (Editores). Resolution of Equations in Algebraic Structures 2. Academic Press. 1989.

[Amo99] J.A. Amor. Compacidad en la Lógica de Primer Orden y su relación con el Teorema de Completud. Facultad de Ciencias UNAM. 1999.

[AM98] J. A. Amor, F. E. Miranda. Demostración Automática de Teoremas y la Nutria. En Memorias del XXX Congreso Nacional de la SMM. Aportaciones Matemáticas 22, Serie Comunicaciones, SMM. México 1998.

[BS94] F. Baader, J. Siekmann. Unification Theory. En [GHR94b] 1994.

[BDP89] L. Bachmair, N. Dershowitz, D. Plaisted. Completion Without Failure. En [AN89] 1989.

[Bra75] D. Brand. Proving Theorems with the Modification Method. SIAM Journal on Computing V.4. No. 4, 1975.

[CL73] C. Chang, R. Lee. Symbolic Logic and Mechanical Theorem Proving. Academic Press. 1973.

[DP60] M. Davis H. Putnam, A Computing Procedure for Cuantification Theory. Journal of the ACM, V.7. No.3, 1960.

[Der89] N. Dershowitz. Completion and Its Applications. En [AN89] 1989.

[GHR94a] D.M. Gabbay, C.J. Hogger, J.A. Robinson (Editores). Handbook of Logic In Artificial Intelligence and Logic Programming, Vol 1. Logical Foundations. Oxford Science Publications. 1994.

[GHR94b] D. M. Gabbay, C.J. Hogger, J.A. Robinson (Editores). Handbook of Logic In Artificial Intelligence and Logic Programming, Vol 2. Deduction Methodologies. Oxford Science Publications. 1994.

[Gil60] P.C. Gilmore. A Proof Method for Cuantification Theory ; its justification and realization. IBM J. Res. Develop. 4, 28-35, 1960.

[Hue81] G. Huet. A complete proof of correctness of the Knuth-Bendix completion algorithm. Journal of Computer and System Sciences. V.23 No. 1. 1981.

[Klo92] J.W. Klop. Term Rewriting Systems. En [AGM92]. 1992.

[KB70] D. Knuth, P. Bendix, Simple Word Problems in Universal Algebras. En Computational Problems in Abstract Algebra. J. Leech, (editor). Oxford Pergamon Press. 1970.

[Kun96] K. Kunnen. The Semantics of Answer Literals. Journal of Automated Reasoning. V. 17. Kluwer. 1996.

[LP91] J.L. Lassez, G. Plotkin (Editores). Computational Logic, Essays in Honor of Alan Robinson. MIT Press. 1991.

[Lei97] A. Leitsch. The Resolution Calculus. Springer Verlag. 1997.

[Lov78] D.W. Loveland. Automated Theorem Proving: A logical basis. North Holland. 1978.

[Llo87] J.W. Lloyd. Foundations of Logic Programming, second edition. Springer-Verlag. 1987.

[McC95] W. McCune. OTTER 3.0 Reference Manual and Guide. Technical Report, UC-405, ANL-94/6. Argonne National Laboratory. 1995.

[McC97] W. McCune. 33 Basic Test Problems: A Practical Evaluation of Some Paramodulation Strategies. En [VP97]. 1997

[MW97] W. McCune, L. Wos. OTTER, The CADE-13 Competition Incarnations. Journal of Automated Reasoning V. 18. Kluwer. 1997.

[Mir99] F. E. Miranda. Lógica Computacional, Notas de Clase para el Curso de Análisis Lógico. Por aparecer, Facultad de Ciencias UNAM 1999.

[Pla94] D. A. Plaisted. Equational Reasoning and Term Rewriting Systems. En [GHR94a]. 1994.

[OL97] R. Overbeek, E. Lusk. Wos and Automated Deduction at ANL: The Ethos. En [VP97]. 1997.

[Qua92] A. Quaife. Automated Development of Fundamental Mathematical Theories. Kluwer Academic Publishers. 1992.

[Ram96] J.A. Ramírez. Un Acercamiento a la Demostración Automática de Teoremas. Tesis de licenciatura. UNAM. 1996.

[RW69] G. Robinson, L. Wos. Paramodulation and theorem proving in first order theories with equality. En Machine Intelligence V. 4. American Elsevier. New York. 1969

[Rob65] J.A. Robinson. A Machine oriented logic based on the resolution principle. Journal of the ACM V. 12 No. 1, 23-41. 1965.

[Sab88] S. Sabre. Traducción Automática a Forma Clausular. Tesis de licenciatura. UNAM. 1988.

[SA91] V. Sperschneider, G. Antoniou. Logic, A Foundation for Computer Science. Addison-Wesley. 1991.

[Sti73] R.B. Stillman. The concept of weak substitution in theorem proving. Journal of the ACM V. 20 No. 4, 648-667. 1973.

[VP97] R. Veroff, G. Pieper (Editores). Automated Reasoning and Its Applications, Essays in Honor of Larry Wos. MIT Press. 1997.

[Wos96] L. Wos. The Automation of Reasoning an Experimenter's Notebook with OTTER Tutorial. Academic Press. 1996.

[Wos98] L. Wos. Programs that Offer Fast, Flawless, Logical Reasoning. Communications of the ACM. V. 41 No. 6. Junio 1998.

[WOL91] L. Wos, R. Overbeek, E. Lusk. Subsumption, a Sometimes Undervaluated Procedure. en [LP91]. 1991.

[WOLB92] L. Wos, R. Overbeek, E. Lusk, J. Boyle. Automated Reasoning, Introduction and Applications. McGraw-Hill. 1992.

[WR70] L. Wos, G. Robinson. Paramodulation and set of support. Proc. Symp. Automatic Demostration. LNM 125, Springer Verlag. 1970.

[WRC65] L. Wos, G. Robinson, D. Carson. Efficiency and completeness of the set of support strategy in theorem proving. Journal of the ACM. V. 12 No. 4. 1965.

[WRC67] L. Wos, G. Robinson, D. Carson. The concept of demodulation in theorem proving. Journal of the ACM. V.14 No. 4. 1967.

[WV94] L. Wos, R. Veroff. Logical basis for the automation of reasoning: case studies. En [GHR94b] 1994.

Índice de Símbolos

Índice