Capítulo 9

Resolución

Al igual que en el tema 6, dedicado a la lógica proposicional, vamos a intentar aquí, dado un conjunto de cláusulas, obtener la cláusula vacía usando resolución. Para eso necesitamos antes el concepto de *resolvente*.

Dadas dos cláusulas C_1 y C_2, una resolvente suya será una nueva cláusula C_3 que se deduzca de las dos dadas, es decir, que se verifique que $\{C_1, C_2\} \vDash C_3$.

9.1. Resolventes

9.1.1. Resolventes binarias

Comenzamos por un ejemplo sencillo.

Supongamos que tenemos las cláusulas $C_1 = \neg P(x) \vee Q(b)$ y $C_2 = P(a)$ y que ambas son ciertas.

La primera cláusula la podemos escribir como $\forall x(P(x) \rightarrow Q(b))$, es decir, que sea quien sea x la fórmula $P(x) \rightarrow Q(b)$ es cierta. En particular, será cierto $P(a) \rightarrow Q(b)$. Como además $P(a)$ es cierto podemos deducir que $Q(b)$ es cierto. Es decir, tenemos que

$$\{\neg P(x) \vee Q(b), P(a)\} \vDash Q(b)$$

Otra forma de llegar a la misma conclusión podría ser:

1. Tomamos las cláusulas C_1 y C_2.
2. Elegimos los literales $L_1 = \neg P(x)$ y $L_2 = P(a)$ de la primera y segunda cláusulas respectivamente.
3. Comprobamos si L_1^C y L_2 son unificables. En este caso lo son, y un unificador principal es $\sigma = (x|a)$.
4. Hallamos $\sigma(C_1) = \neg P(a) \vee Q(b)$ y $\sigma(C_2) = P(a)$
5. Eliminamos de $\sigma(C_1)$ y $\sigma(C_2)$ los literales $\sigma(L_1)$ y $\sigma(L_2)$, y formamos una cláusula con los literales restantes (aquí procedemos de igual forma a como hacíamos en el tema sexto cuando calculábamos resolventes). La cláusula resultante es consecuencia lógica de las dos primeras.

Vamos a repetir este proceso con las cláusulas

$$C_1 = P(x, b) \vee Q(x, a); \qquad C_2 = \neg P(a, z) \vee R(z)$$

1. Tomamos el literal $L_1 = P(x, b)$ de la primera cláusula, y el literal $L_2 = \neg P(a, z)$ de la segunda.
2. Buscamos un unificador principal para L_1^C y L_2. Este unificador existe, y es $\sigma = (x|a; z|b)$.
3. Calculamos $\sigma(C_1) = P(a, b) \vee Q(a, a)$ y $\sigma(C_2) = \neg P(a, b) \vee R(b)$.
4. Eliminamos los literales $\sigma(L_1)$ y $\sigma(L_2)$ de las cláusulas $\sigma(C_1)$ y $\sigma(C_2)$. Nos queda la cláusula $C_3 = Q(a, a) \vee R(b)$

Vamos a ver que $\{C_1, C_2\} \vDash C_3$.

Suponemos que $I(C_1) = I(C_2) = 1$.

Entonces $I(\forall x(P(x,b) \vee Q(x,a))) = 1$. Significa esto que $P(x,b) \vee Q(x,a)$ será cierto sea cual sea el valor de x. En particular, $I(P(a,b) \vee Q(a,a)) = 1$.

De la misma forma, como $I(C_2) = 1$, entonces $I(\neg P(a,b) \vee R(b)) = 1$.

Pueden ocurrir ahora dos cosas:

$I(P(a,b)) = 1$

En este caso $I(\neg P(a,b)) = 0$, luego $I(R(b)) = 1$ (ya que $I(\neg P(a,b) \vee R(b)) = 1$). Por tanto, $I(Q(a,a) \vee R(b)) = I(C_3) = 1$.

$I(P(a,b)) = 0$

En este caso $I(Q(a,a))$ debe valer uno, luego $I(C_3) = 1$.

Luego en cualquiera de los casos deducimos que $I(C_3) = 1$, es decir,

$$\{C_1, C_2\} \vDash C_3.$$

Notemos que la cláusula C_2 es equivalente a $\forall x(\neg P(a,x) \vee R(x))$. Pero ahora, cuando intentamos buscar un unificador para $P(x,b)$ y $P(a,x)$ vemos que no existe, luego no podríamos seguir el proceso anterior. Hay que tener en cuenta que **las variables de dos cláusulas distintas son distintas, aunque tengan el mismo nombre. Caso de que coincidan los nombres es conveniente cambiar el nombre de las variables de alguna de ellas.**

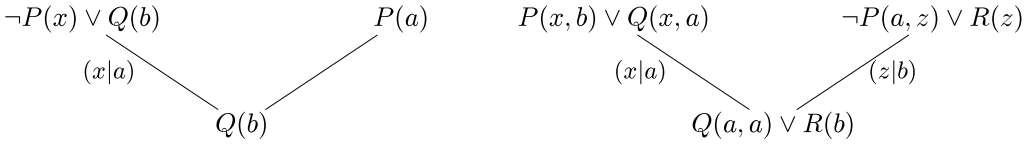
Definimos ya lo que es una resolvente binaria de dos cláusulas:

Definición 65. *Sean C_1 y C_2 dos cláusulas que no tienen variables comunes. Supongamos que $C_i = L_i \vee C_i'$, donde L_i, $i = 1,2$ es un literal y $C_i, i = 1,2$ es una cláusula (que podría ser la vacía), y que los literales L_1 y L_2^C tienen un unificador principal σ.*

Entonces la cláusula $\sigma(C_1') \vee \sigma(C_2')$ es una resolvente binaria de C_1 y C_2.

Por ejemplo, $Q(b)$ es una resolvente binaria de las cláusulas $\neg P(x) \vee Q(b)$ y $P(a)$, mientras que $Q(a,a) \vee R(b)$ es una resolvente binaria de $P(x,b) \vee Q(x,a)$ y $\neg P(a,z) \vee R(z)$.

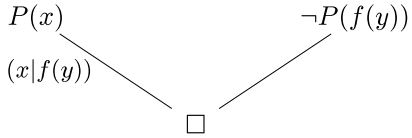
Normalmente, para indicar que una cláusula se obtiene como resolvente binaria de otras dos usaremos la notación siguiente:



Una vez visto cómo obtener una resolvente, podemos afirmar que si de un conjunto de cláusulas podemos obtener la cláusula vacía haciendo resolventes, entonces el conjunto es insatisfacible.

Por ejemplo, podemos ver que el conjunto $\{P(x), \neg P(f(x))\}$ es insatisfacible.

En principio podría parecer que no es posible obtener ninguna resolvente, pues $P(x)$ y $P(f(x))$ no son unificables. Sin embargo, hemos dicho que cuando tengamos variables que aparecen en dos cláusulas, podemos cambiar las de una de ellas. En este caso nos quedaría el conjunto de cláusulas como $\{P(x), \neg P(f(y))\}$, y ahora si podemos hacer resolventes:



Tomamos el conjunto $\{P(x) \vee P(y), \neg P(z) \vee \neg P(u)\}$. Este es también insatisfacible y sin embargo no podemos obtener de ninguna forma la cláusula vacía haciendo resolventes binarias.

Necesitamos por tanto extender el concepto de resolvente para que podamos contemplar casos como este (y otros algo más complicados).

9.1.2. Resolventes

Para definir una resolvente, necesitamos previamente el concepto de factor.

Sea C una cláusula. Supongamos que en C hay dos o más literales que son unificables, y sea σ un unificador principal. En ese caso, diremos que $\sigma(C)$ es un **factor** de C. En el caso de que $\sigma(C)$ sea un factor de C se verifica que $C \vDash \sigma(C)$.

Por ejemplo, si $C = P(x) \vee P(f(a)) \vee Q(x, b)$ entonces $P(f(a)) \vee Q(f(a), b)$ es un factor de C (hemos unificado los dos primeros literales con la sustitución $(x|f(a))$).

Notemos que si queremos hallar un factor de una cláusula, y en dos o más literales tenemos variables repetidas, para obtener un unificador **no podemos** renombrar las variables (salvo que a todas las apariciones de la variable en la cláusula le demos el mismo nombre).

Definición 66. *Sean C_1 y C_2 dos cláusulas. Se dice que C es una* **resolvente** *de C_1 y C_2 si C responde a alguna de las cuatro posibilidades siguientes:*

1. *C es una resolvente binaria de C_1 y C_2.*
2. *C es una resolvente binaria de C_1 y un factor de C_2.*
3. *C es una resolvente binaria de un factor de C_1 y de C_2.*
4. *C es una resolvente binaria de un factor de C_1 y un factor de C_2.*

Notemos que si C es una resolvente de C_1 y C_2 entonces $\{C_1, C_2\} \vDash C$.

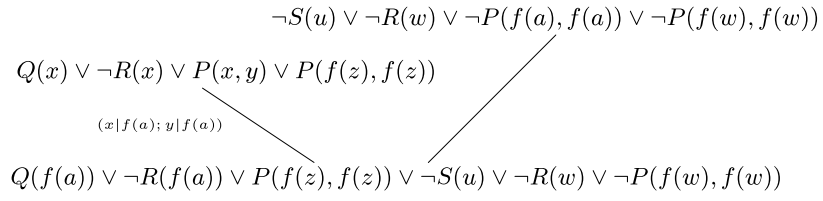
Ejemplo 9.1.1.

1. *Consideramos las cláusulas $C_1 = P(x) \vee P(y)$ y $C_2 = \neg P(z) \vee \neg P(u)$. Entonces $P(x)$ es un factor de C_1, $\neg P(z)$ es un factor de C_2, y $\square$ es una resolvente binaria de $P(x)$ y $\neg P(z)$. Por tanto, $\square$ es una resolvente de C_1 y C_2.*

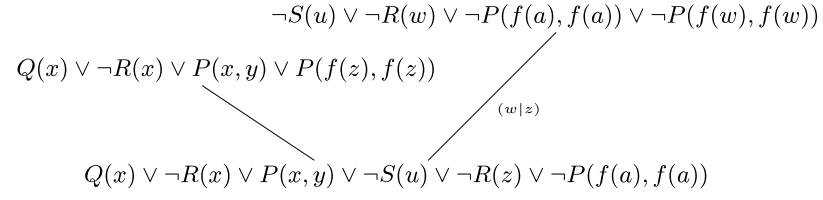
   Vemos así que el conjunto $\{P(x) \vee P(y), \neg P(z) \vee \neg P(u)\}$ es insatisfacible.

2. *Vamos ahora a obtener varias resolventes de las cláusulas*

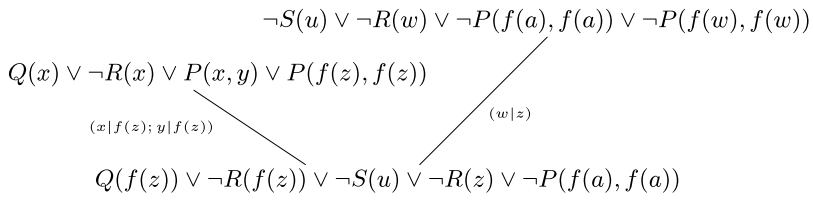
$$C_1 = Q(x) \vee \neg R(x) \vee P(x, y) \vee P(f(z), f(z))$$
$$C_2 = \neg S(u) \vee \neg R(w) \vee \neg P(f(a), f(a)) \vee \neg P(f(w), f(w))$$



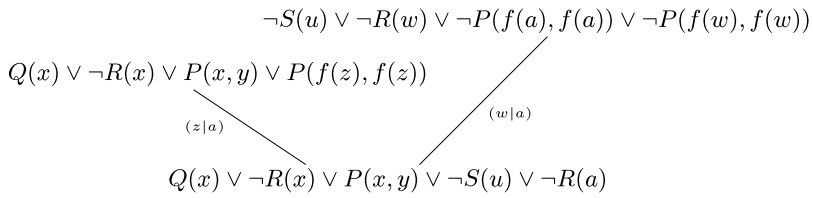
En este caso se ha obtenido una resovente binaria de C_1 y C_2.



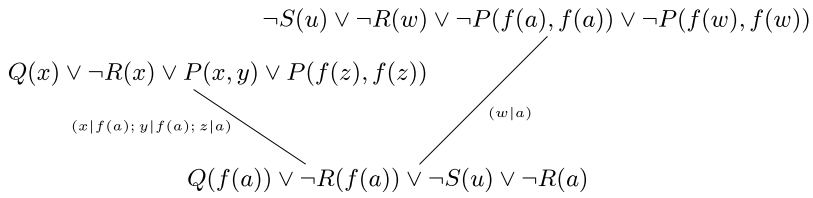
En este caso, también hemos obtenido una resolvente binaria de C_1 y C_2. Podemos conseguir dos más.



Aquí hemos hecho una resolvente binaria de un factor de C_1 y C_2. Podríamos hacer otra más.



Aquí, una resolvente binaria de C_1 y un factor de C_2. Se puede hacer otra más.



Por último, lo que hemos hecho es una resolvente binaria de un factor de C_1 y un factor de C_2.

9.1.3. Deducciones y principio de resolución.

Sea Γ un conjunto de cláusulas, y C una cláusula. Una *deducción* de C a partir de Γ es una sucesión finita de cláusulas $C_1, C_2, \cdots C_n$ donde $C_n = C$, y para $i < n$, C_i es una cláusula, que bien es un elemento de Γ, bien es una resolvente de dos cláusulas que la preceden.

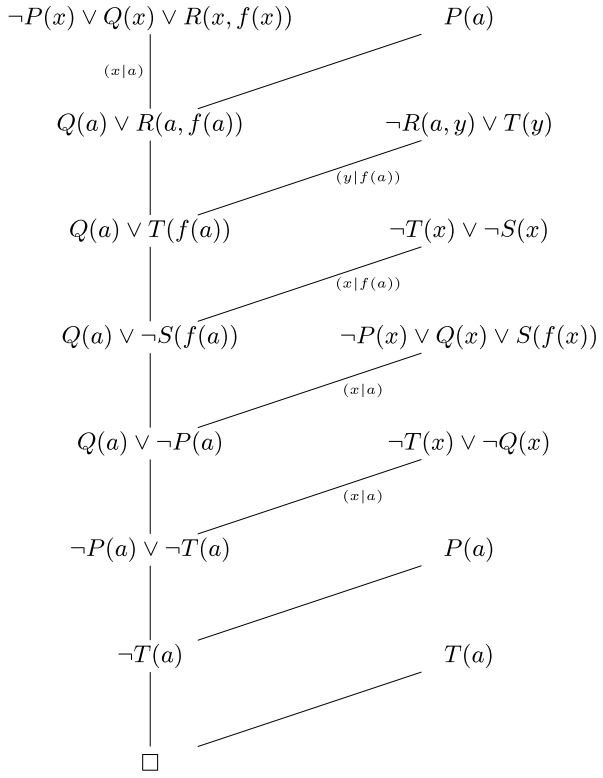
Ejemplo 9.1.2. *En el ejemplo 7.1.10 estuvimos viendo que para comprobar si*

$$\left\{ \begin{array}{l} \forall x(P(x) \wedge \neg Q(x) \rightarrow \exists y(R(x,y) \wedge S(y))) \\ \exists y \forall x((R(y,x) \rightarrow T(x)) \wedge T(y) \wedge P(y) \end{array} \right\} \models \exists x(T(x) \wedge (Q(x) \vee S(x)))$$

bastaba con comprobar si el conjunto de cláusulas

$$\Gamma = \left\{ \begin{array}{llll} \neg P(x) \vee Q(x) \vee R(x, f(x)); & \neg R(a,y) \vee T(y); & T(a); & P(a) \\ \neg P(x) \vee Q(x) \vee S(f(x)); & \neg T(x) \vee \neg Q(x); & \neg T(x) \vee \neg S(x) & \end{array} \right\}$$

es insatisfacible. Ahora vamos a ver que este conjunto es insatisfacible, para lo cual vamos a dar una deducción de la cláusula vacía.



Si consideramos las cláusulas:

$$C_1 = \neg P(x) \vee Q(x) \vee R(x, f(x)); C_2 = P(a); C_3 = Q(a) \vee R(a, f(a)); C_4 = \neg R(a, y) \vee T(y);$$
$$C_5 = Q(a) \vee T(f(a)); C_6 = \neg T(x) \vee \neg S(x); C_7 = Q(a) \vee \neg S(f(a)); C_8 = \neg P(x) \vee Q(x) \vee S(f(x));$$
$$C_9 = Q(a) \vee \neg P(a); C_{10} = \neg T(x) \vee \neg Q(x); C_{11} = \neg P(a) \vee \neg T(a); C_{12} = \neg T(a); C_{13} = T(a); C_{14} = \square$$

entonces:

1. C_1 *y* C_2 *son elementos de* Γ.
2. C_3 *es resolvente de* C_1 *y* C_2.
3. C_4 *es un elemento de* Γ.
4. C_5 *es una resolvente de* C_3 *y* C_4.
5. C_6 *es un elemento de* Γ.
6. C_7 *es una resolvente de* C_5 *y* C_6.
7. C_8 *es un elemento de* Γ.
8. C_9 *es una resolvente de* C_7 *y* C_8.
9. C_{10} *es un elemento de* Γ.
10. C_{11} *es una resolvente de* C_9 *y* C_{10}.
11. C_{12} *es una resolvente de* C_2 *y* C_{11}.
12. C_{13} *es un elemento de* Γ.
13. C_{14} *es una resolvente de* C_{12} *y* C_{13}.

Por tanto $C_{14} = \square$ *se ha deducido a partir de* Γ.

Una vez visto esto podemos dar el teorema principal de este capítulo:

Teorema 9.1.1. *(Completitud del principio de resolución)*

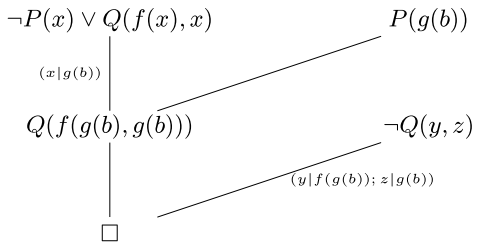
Sea Γ *un conjunto de cláusulas. Entonces* Γ *es insatisfacible si, y sólo si, existe una deducción de* $\square$ *a partir de* Γ.

Ejemplo 9.1.3.

1. *Vamos a comprobar que el conjunto de cláusulas*

$$\Gamma = \{\neg P(x) \vee Q(f(x), x), \quad P(g(b)), \quad \neg Q(y, z)\}$$

es insatisfacible. Para esto, vamos a obtener una deducción de la cláusula vacía.



2. *Vamos a comprobar que*

$$\{\forall x Q(x) \vee \exists y P(y),\ \neg\forall y Q(y)\} \vDash \exists x P(x)$$

Demostrar eso es lo mismo que probar que el conjunto de fórmulas

$$\{\forall x Q(x) \vee \exists y P(y),\ \neg\forall y Q(y),\ \neg\exists x P(x)\}$$

es insatisfacible.

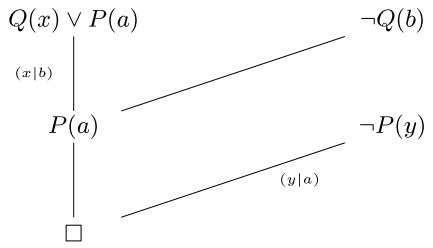
Para esto, hallamos la forma clausular de cada una de las fórmulas:

$$\left|\begin{array}{c|c|c} \forall x Q(x) \vee \exists y P(y) & \neg\forall y Q(y) & \neg\exists x P(x) \\ \exists y \forall x (Q(x) \vee P(y)) & \exists y \neg Q(y) & \forall x \neg P(x) \\ \forall x (Q(x) \vee P(a)) & \neg Q(b) & \forall y \neg P(y) \end{array}\right|$$

Luego debemos ver que el conjunto de cláusulas

$$\{Q(x) \vee P(a),\ \neg Q(b),\ \neg P(y)\}$$

es insatisfacible.



9.2. Estrategias de gestión

Podría parecer que con el Teorema de Completitud del Principio de Resolución está resuelto el problema de la insatisfacibilidad de un conjunto de cláusulas, y por tanto el problema de la implicación semántica. Sin embargo, esto está muy lejos de ser cierto.

Sabemos que si un conjunto de cláusulas es insatisfacible, entonces existe una deducción (que contiene un número finito de pasos) de la cláusula vacía. Pero no existe ningún algoritmo que nos dé la respuesta a la pregunta de si un conjunto es o no insatisfacible, aunque, si es insatisfacible, hay algoritmos que nos conducen a la cláusula vacía en un número finito de pasos (otro tema es que eso puede ser computacionalmente muy costoso). La dificultad está en que si el conjunto de cláusulas es satisfacible no hay ningún algoritmo que nos dé la respuesta en un finito de pasos.

Lo que vamos a ver a continuación son distintas estrategias para, dado un conjunto de cláusulas, obtener, si es insatisfacible, una deducción de la cláusula vacía.

9.2.1. Estrategia de Saturación

Recordemos que el problema que tenemos en este momento es el de determinar si un conjunto de cláusulas es satisfacible o insatisfatisfacible. La búsqueda de una deducción de la cláusula vacía (también llamada una refutación) o llegar a la conclusión de que ésta no existe puede abordarse calculando todas las posibles resolventes a partir del conjunto de partida. Es la **estrategia de saturación**. El procedimiento puede ejecutarse de una forma algorítmica que describimos a continuación:

Llamamos $S_0 = S$.

Para cada i

> Calculamos S_{i+1} como el conjunto que se obtiene de S_i al añadir todas las resolventes que puedan calcularse usando cláusulas de S_i.
>
> Si S_{i+1} contiene a la cláusula vacía, entonces el conjunto de partida es **insatisfacible**;
>
> si $S_{i+1} = S_i$, entonces el conjunto de partida es **satisfacible**;
>
> en otro caso incrementar i y volver a ejecutar el bucle.

Este algoritmo nos proporciona una cadena de conjuntos de cláusulas, es decir, una secuencia de conjuntos en el que cada uno está contenido en el siguiente:

$$S_0 \subseteq S_1 \subseteq \cdots \subseteq S_i \subseteq S_{i+1} \subseteq \ldots$$

y de forma que todos tienen el mismo carácter de satisfacibilidad (es decir, uno es insatisfacible si, y sólo si, lo es otro cualquiera).

Cuando en uno de los eslabones aparece la cláusula vacía tenemos asegurada la insatisfacibilidad; cuando la cadena se estabiliza (es decir $S_i = S_{i+1}$, y por tanto todos los siguientes vuelven a ser iguales a S_i puesto que no aparecen nuevas resolventes) entonces ya se tiene que no es posible obtener una deducción de la cláusula vacía, puesto que se han explorado **todas** las posibilidades. Hay dos problemas que presenta este método: la imposibilidad de detener el proceso en un número finito de pasos y que el esfuerzo de cálculo sea inabordable.

Ejemplo 9.2.1. *Consideremos el conjunto*

$$S = \{\neg P(x) \vee Q(f(x)),\ P(a),\ \neg P(y) \vee \neg Q(y)\}$$

Inicializamos $S_0 = S$ y calculamos las posibles resolventes entre sus cláusulas. Las numeramos para obtener una más fácil referencia:

$C_1 = \neg P(x) \vee Q(f(x))$

$C_2 = P(a)$

$C_3 = \neg P(x) \vee \neg Q(x)$

entonces podemos obtener las siguientes nuevas cláusulas:

$C_4 = R(C_1, C_2) = Q(f(a))$ *con el unificador* $(x|a)$*;*

$C_5 = R(C_1, C_3) = \neg P(x) \vee \neg P(f(x))$ *con el unificador* $(y|f(x))$*;*

$C_6 = R(C_2, C_3) = \neg Q(a)$ *con el unificador* $(x|a)$*.*

El conjunto S_1 *consta de las seis cláusulas que tenemos hasta el momento:*

$$S_1 = \{\neg P(x) \vee Q(f(x)),\ P(a),\ \neg P(y) \vee \neg Q(y),\ Q(f(a)),\ \neg P(x) \vee \neg P(f(x)),\ \neg Q(a)\}$$

Como no contiene a la cláusula vacía ni coincide con el anterior, entonces proseguimos calculando S_2*:*

A partir de la C_1 *no es posible calcular nuevas resolventes;*

$C_7 = R(C_2, C_5) = R(C_3, C_4) = \neg P(f(a))$*;*

tampoco hay más resolventes entre el resto de cláusulas.

$$S_2 = \left\{ \begin{array}{llll} \neg P(x) \vee Q(f(x)); & P(a); & \neg P(y) \vee \neg Q(y); & Q(f(a)); \\ \neg P(x) \vee \neg P(f(x)); & \neg Q(a); & \neg P(f(a)) & \end{array} \right\}$$

De nuevo examinamos si aparece la cláusula vacía, lo que no ocurre, y $S_2 \neq S_1$*, así que debemos continuar calculando* S_3*: como sólo hay una cláusula nueva entonces las nuevas resolventes tendrían que serlo de* C_7*, pero ésta no admite ninguna resolvente con el resto. Así que*

$$S_3 = S_2$$

y el algoritmo acaba emitiendo la respuesta: **el conjunto es satisfacible**.

Ejemplo 9.2.2. *Sea ahora* $S_0 = \{A(b),\ \neg M(y) \vee P(b, y),\ \neg P(x, z),\ M(a),\ C(a)\}$ *y comencemos a ejecutar el algoritmo:*

$C_6 = R(C_2, C_3) = \neg M(y)$

$C_7 = R(C_2, C_4) = P(b, a)$

Así que S_1 *consta de las siete cláusulas descritas, no contiene a la cláusula vacía y tampoco coincide con el anterior. Calculamos* S_2 *y encontramos* $C_8 = R(C_4, C_6) = \square$ *con lo que el algoritmo acaba con la respuesta* **el conjunto es insatisfacible**. *Para reconstruir la deducción que nos lleva a la cláusula vacía recorremos el proceso en sentido inverso:*

$\square = R(C_4, C_6)$

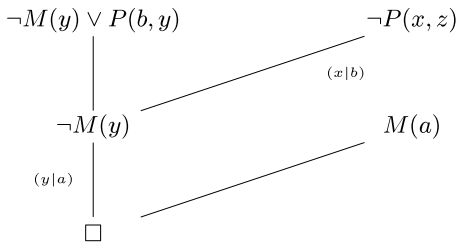
$C_6 = R(C_2, C_3)$

C_2 *y* C_3 *son cláusulas del conjunto de partida.*

y la deducción sería:

1. C_2 *está en* S*;*
2. C_3 *está en* S*;*
3. C_6 *es resolvente de las dos anteriores;*
4. $\square$ *es resolvente de dos anteriores.*

o dibujada en forma de árbol:



Los dos ejemplos anteriores son representativos de la situación habitual, cuando la cantidad de cláusulas que aparecen en los sucesivos conjuntos S_i pueden ser enormes. Se propone como ejercicio realizar los primeros pasos de la estrategia de saturación para el siguiente conjunto de cláusulas:

$$\Gamma = \{\neg E(x,y) \vee \neg E(x,z) \vee E(z,y),\ \neg E(u,v) \vee E(v,u),\ E(a,b),\ E(b,c),\ \neg E(a,c)\ \}$$

Por último, tomamos el conjunto de cláusulas

$$\Gamma = \{\neg P(x) \vee P(f(x));\ P(a)\}$$

Partimos de $S_0 = \Gamma$, y vamos construyendo los distintos conjuntos S_i:

$$S_1 = \{\neg P(x) \vee P(f(x));\ P(a);\ P(f(a))\}$$
$$S_2 = \{\neg P(x) \vee P(f(x));\ P(a);\ P(f(a));\ P(f(f(a)))\}$$
$$S_3 = \{\neg P(x) \vee P(f(x));\ P(a);\ P(f(a));\ P(f(f(a)));\ P(f(f(f(a))))\}$$

Y podemos ver como obtenemos una sucesión

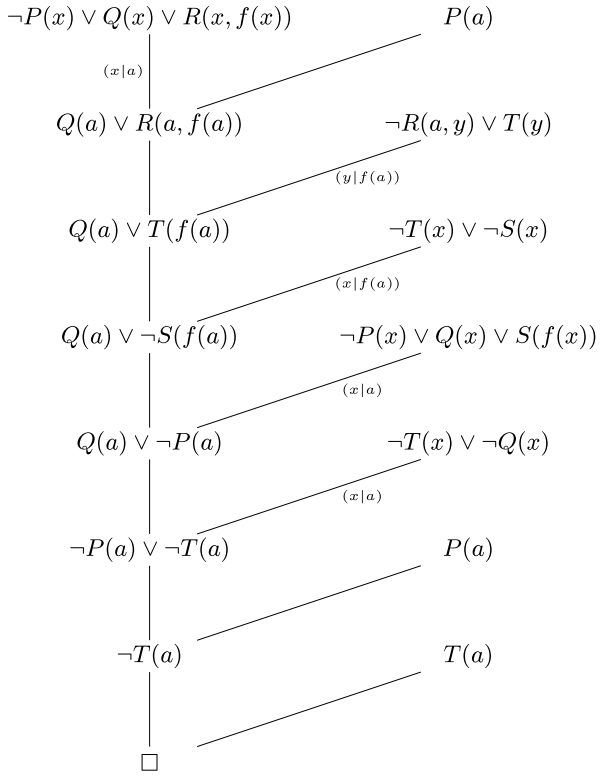
$$S_0 \subsetneq S_1 \subsetneq S_2 \subsetneq S_3 \subsetneq \cdots \subsetneq S_i \subsetneq S_{i+1} \subsetneq \cdots$$

y en la que nunca nos aparecerá la cláusula vacía.

Por tanto, el algoritmo en este caso no terminaría. Si en algún caso no llegamos a ninguna de las dos condiciones de parada, ¿cuándo terminamos?. Si no nos ha salido la cláusula vacía, ¿es porque no se puede obtener, o porque no hemos hecho las iteraciones necesarias para conseguirla?.

9.2.2. Deducciones lineales

Hasta ahora hemos escrito y dibujado varios ejemplos de deducciones. Si observamos el esquema de la deducción:

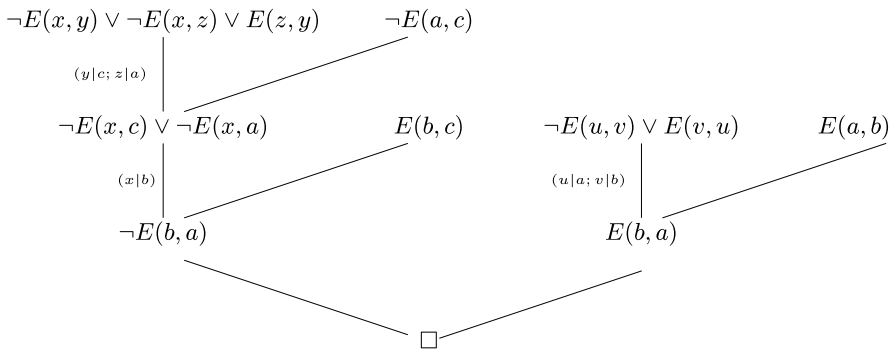


notamos que cada nueva resolvente se ha calculado utilizando justo la que se ha obtenido en el paso anterior. En el dibujo aparece una línea vertical que nos lleva desde la cláusula de partida a la cláusula vacía. Es lo que se llama una **deducción lineal**. No todas las deducciones tienen que verificar esta propiedad:

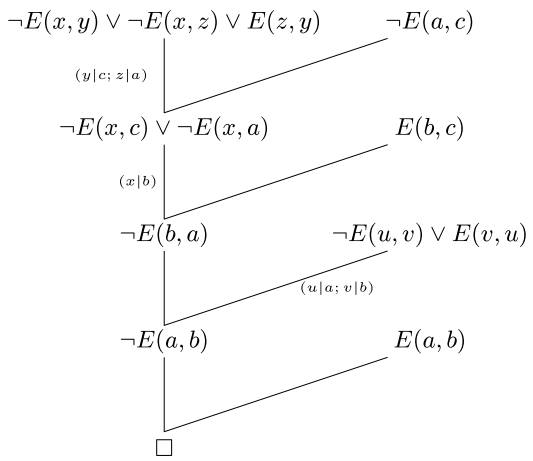
Ejemplo 9.2.3. *Veamos el dibujo de una refutación para el conjunto*

$$\Gamma = \{\neg E(x,y) \vee \neg E(x,z) \vee E(z,y),\ \neg E(u,v) \vee E(v,u),\ E(a,b),\ E(b,c),\ \neg E(a,c)\}$$

que no es lineal:



Sin embargo, a partir de cada deducción se puede obtener una que sea **lineal**. *En este caso podemos modificar la anterior así:*



Esto es un resultado general, se tiene:

Teorema 9.2.1. *Si para un conjunto de cláusulas existe una deducción de la cláusula vacía, entonces existe una deducción lineal de la cláusula vacía.*

Y como consecuencia en adelante **nos limitaremos a usar deducciones lineales** (o refutaciones lineales).

9.2.3. Deducciones lineales-input

Pero el problema principal que teníamos era el de la gran cantidad de posibilidades para calcular resolventes, que pueden convertir un problema en inabordable en la práctica. Existen distintas opciones de restricción a la hora de elegir resolventes a calcular. En primer lugar observemos que podríamos limitarnos a las deducciones lineales que parten de **una cláusula fija** a la que llamaremos **raíz**. En el deducción lineal del ejemplo anterior se ha usado $\neg E(x,y) \vee \neg E(x,z) \vee E(z,y)$ como raíz de la deducción. Si además limitásemos las posibles cláusulas que pueden entrar para calcular resolventes en la línea, tendríamos un número de posibilidades pequeño. Al conjunto de cláusulas que se elige para entrar a formar resolventes se le suele llamar **conjunto usable**.

Cuando elegimos una raíz y un conjunto usable, podemos dibujar el conjunto de todas las deducciones lineales que parten de la raíz como un árbol en el que en cada nodo aparecen tantas ramas como resolventes podamos hacer con esa cláusula y las del conjunto usable. Cada rama es una deducción lineal, luego debemos buscar una rama que acabe en la cláusula vacía.

El problema es que estas restricciones pueden hacer que la deducción lineal que nos lleva hasta la cláusula vacía no nos parezca entre las calculadas.

Ejemplo 9.2.4. *Para el conjunto*

$$\Gamma = \{\neg E(x,y) \vee \neg E(x,z) \vee E(z,y),\ \neg E(u,v) \vee E(v,u),\ E(a,b),\ E(b,c),\ \neg E(a,c)\}$$

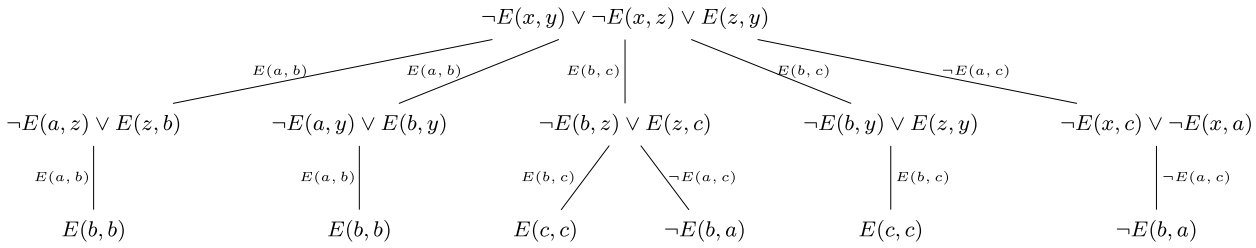
elegimos como raíz la cláusula

$$\neg E(x,y) \vee \neg E(x,z) \vee E(z,y)$$

y tomamos como conjunto usable el de las cláusulas del conjunto de partida que constan de un solo literal (es una estrategia de selección llamada **UNIT***), es decir*

$$\begin{cases} C_3 = E(a,b) \\ C_4 = E(b,c) \\ C_5 = \neg E(a,c) \end{cases}$$

Dibujamos el árbol de **todas** *las deducciones lineales:*



En las hojas que nos quedan no pueden calcularse nuevas resolventes con el conjunto usable que hemos elegido $\{C_3, C_4, C_5\}$*, y en ninguna de las ramas del árbol aparece la cláusula vacía, por lo que hemos perdido la solución adecuada del problema.*

En el ejemplo anterior una posible solución es permitir que el conjunto usable se alimente también de las cláusulas unit que se vayan obteniendo como resolvente a lo largo del proceso. Cuando no se permite la introducción de nuevas cláusulas a lo largo del proceso en el conjunto usable decimos que la estrategia es **input**.

Ejemplo 9.2.5. *Para otros conjuntos de cláusulas la estrategia puede tener un resultado satisfactorio. Por ejemplo, si tomamos el conjunto*

$$\Gamma = \{\neg P(x) \vee Q(x),\ \neg P(x) \vee \neg Q(x),\ P(a),\ Q(b)\}$$

la estrategia de obtener las deducciones lineales que parten de la raíz $\neg P(x) \vee \neg Q(x)$ *y utilizar como conjunto usable el de las cláusulas unit da una respuesta rápidamente. En cambio, con la raíz* $\neg P(x) \vee Q(x)$ *no aparece una solución. Eso significa que la elección de la raíz es esencial en la obtención de una solución.*

La estrategia de selección de deducciones que vamos a utilizar es la de búsqueda de deducciones lineales que son input (esto es, las resolventes que se obtienen **no** se introducen en el conjunto usable) y con conjunto usable todas las cláusulas de partida salvo la que se elige como raíz: son las deducciones **lineales-input**.

Resumimos las características de esta estrategia de elección de deducciones:

- Se elige una cláusula raíz.
- El resto de las cláusulas del conjunto de partida forman el conjunto usable.
- Se consideran deducciones lineales.

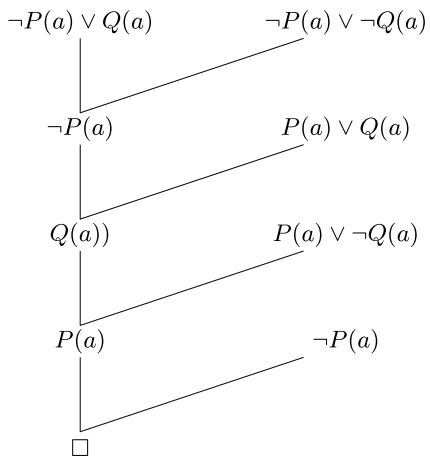
Así podremos construir el **árbol de las deducciones lineales-input** que tendrá por nodo raíz a la cláusula que hayamos elegido. Una deducción lineal-input de la cláusula vacía es un recorrido desde la raíz del árbol a una rama que termine en la cláusula vacía.

Esta estrategia de elección de deducciones lineales-input **no es completa**, es decir, puede que el conjunto sea insatisfacible y no exista ninguna deducción lineal-input.

Ejemplo 9.2.6. *Consideremos el conjunto*

$$\{\neg P(a) \vee Q(a),\ \neg P(a) \vee \neg Q(a),\ P(a) \vee Q(a),\ P(a) \vee \neg Q(a)\}$$

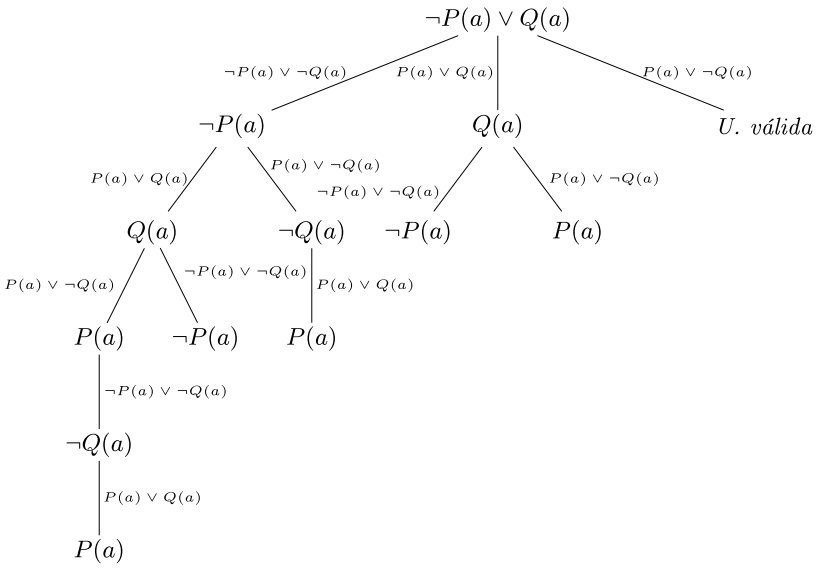
que es insatisfacible y que tiene como refutación lineal:



que **no es input** *puesto que en la última resolvente participa una cláusula que se ha obtenido en el proceso ($\neg P(a)$).*

Sin embargo el árbol de las deducciones lineales-input (con cualquier elección de la raíz) no tiene como nodo a la clásula vacía. Podemos intentar construir el que tiene parte de $\neg P(a) \vee Q(a)$; para ello ordenamos el resto de cláusulas y construimos el árbol calculando resolventes en el orden en el que se presentan el conjunto usable:

$$\begin{cases} C_1 = \neg P(a) \vee \neg Q(a), \\ C_2 = P(a) \vee Q(a), \\ C_3 = P(a) \vee \neg Q(a) \end{cases}$$



Y vemos como cualquier rama que pase por $P(a)$, como cualquier rama que pase por $\neg P(a)$ puede prolongarse hasta el infinito, y nunca llega a la cláusula vacía.

Así que el árbol de todas las deducciones lineales-input no contiene a la cláusula vacía, aunque sabemos que el conjunto es insatisfacible.

Si hubiéramos tomado como raíz otra de las cláusulas, nos saldría un árbol de deducciones semejante al que hemos obtenido aquí.

9.2.4. Recorridos en el árbol de las deducciones

Incluso en el caso en que la cláusula vacía aparezca en el árbol de las deducciones lineales-input, a la hora de acometer de forma algorítmica su búsqueda podemos tener problemas prácticos. Suponemos que son conocidas técnicas de recorridos de árboles, en concreto los recorridos primero en anchura y primero en profundidad. Un recorrido explorando primero en profundidad puede encontrar una rama infinita que nos haga imposible el acceso a otras ramas; el recorrido primero en anchura encuentra con seguridad la cláusula vacía si ésta se encuentra entre los nodos, el inconveniente es que los requisitos de memoria de este tipo de exploración puede ser tremendamente costosos. En las implementaciones suelen adoptarse métodos de exploración primero en profundidad pero que contemplen una interrupción forzada después de un determinado número de pasos.

9.3. Conjuntos de Horn

En la sección anterior aparecía un ejemplo de conjunto insatisfacible en el que el árbol de las deducciones lineales-input con una raiz determinada no contiene a la cláusula vacía; es decir, la estrategia de reducirse a las deducciones lineales-input no es una estrategia completa en el caso general. Sin embargo la focalización hacia ella que hemos realizado en la sección anterior está totalmente justificada por su importancia en los casos que se ajustan al modelo que presentamos a continuación.

9.3.1. Descripción de un conjunto de Horn

Definición 67. *Una cláusula se dice que es una* **cláusula de Horn** *si tiene exactamente un literal positivo.*

Entre las cláusulas de Horn podemos distinguir dos tipos según si tienen algún literal negativo, las llamadas **reglas**, o si sólo contienen al literal positivo, los **hechos**.

Ejemplo 9.3.1. *La siguiente es una lista de cláusulas de Horn:*

$C(b)$ *es un hecho;*

$D(x) \vee \neg M(x)$ *es una regla;*

$\neg D(x) \vee M(x)$ *es una regla;*

$\neg C(y) \vee CC(f(y), y)$ *es una regla;*

$\neg C(y) \vee \neg CC(x, y) \vee M(x)$ *es una regla.*

En términos del lenguaje de programación **PROLOG** un conjunto de reglas y hechos, es decir, un conjunto de cláusulas de Horn, forman un **programa**.

Cuando una cláusula tiene todos sus literales negativos la llamaremos **cláusula negativa u objetivo**. Un ejemplo de cláusula negativa es

$$\neg M(x) \vee \neg D(x) \vee \neg CC(x, y) \vee \neg C(y)$$

Definición 68. *Un conjunto de cláusulas se dice que es un* **conjunto de Horn** *si contiene exactamente una cláusula negativa y el resto son cláusulas de Horn.*

Ejemplo 9.3.2. *El conjunto de cláusulas*

$$\{C(b);\ D(x) \vee \neg M(x);\ \neg D(x) \vee M(x);\ \neg C(y) \vee CC(f(y), y);$$
$$\neg C(y) \vee \neg CC(x, y) \vee M(x);\ \neg M(x) \neg D(x) \vee \neg CC(x, y) \neg C(y)\}$$

es un conjunto de Horn.

El resultado que nos permite reducirnos a las deducciones lineales-input en el caso de los conjuntos de Horn es el siguiente:

Teorema 9.3.1. *Si Γ es un conjunto de Horn insatisfacible entonces existe una deducción de la cláusula vacía que es lineal-input y que parte de la cláusula objetivo.*

Es decir, la estrategia de reducirnos a las deducciones lineales-input con raíz la cláusula objetivo (negativa) y conjunto usable el formado por las cláusula de Horn, es completa. Así, en estas circunstancias, será suficiente explorar el árbol de las deducciones lineales-input con raíz el objetivo para determinar si el conjunto es o no insatisfacible. Como ya advertimos en la sección anterior, otro problema es si esta exploración se puede ejecutar de una forma satisfactoria.

9.3.2. Resolventes en conjuntos de Horn

Estamos ahora en la situación de calcular deducciones input- lineales partiendo de la cláusula objetivo. Queremos poner de manifiesto qué forma tienen las resolventes que se calculan en cada paso y para ello vamos a trabajar sobre el ejemplo de conjunto de Horn del apartado anterior:

$$\{C(b);\ D(x) \vee \neg M(x);\ \neg D(x) \vee M(x);\ \neg C(y) \vee CC(f(y), y);$$
$$\neg C(y) \vee \neg CC(x, y) \vee M(x);\ \neg M(x)\neg D(x) \vee \neg CC(x, y)\neg C(y)\}$$

La cláusula negativa u **objetivo** es

$$\neg M(x) \vee \neg D(x) \vee \neg CC(x, y) \vee \neg C(y)$$

y el conjunto usable son las cláusulas de Horn en las que hemos situado en primera posición siempre el literal positivo:

1. $C(b)$
2. $D(x) \vee \neg M(x)$
3. $M(x) \vee \neg D(x)$
4. $CC(f(y), y) \vee \neg C(y)$
5. $M(x) \vee \neg C(y) \vee \neg CC(x, y)$

Comencemos calculando la resolvente de la raíz con la primera cláusula que es un **hecho**: unificamos los literales $C(b)$ y $C(y)$ con $(y|b)$ y la resolvente queda:

$$\neg M(x) \vee \neg D(x) \vee \neg CC(x, b)$$

que es **de nuevo una cláusula objetivo**, es decir tiene todos sus literales negativos. Lo mismo ocurre si calculamos resolvente con una regla, veamos una pequeña justificación general:

Tengamos $\neg P_1 \vee \neg P_2 \cdots \vee \neg P_s$ una cláusula objetivo, y $P_1 \vee \neg Q_1 \cdots \vee \neg Q_r$ una regla. Entonces la resolvente (observemos que sólo se puede resolver con el literal positivo de la cláusula de Horn) sería

$$\neg P_2 \cdots \vee \neg P_s \vee \neg Q_1 \cdots \vee \neg Q_r$$

que vuelve a ser una cláusula objetivo.

Así, cuando se calcula el árbol de las deducciones lineales-input en cada nodo se tiene siempre una cláusula negativa u objetivo desde la que se calcularán resolventes con aquellas cláusulas de Horn que tengan el literal positivo unificable con alguno de los del objetivo.

Ejemplo 9.3.3. *Partiendo de la raiz $\neg M(x)\neg D(x) \vee \neg CC(x, y) \vee \neg C(y)$ podemos calcular 5 resolventes distintas, esto es, se abren 5 ramas distintas del árbol que organizaríamos de izquierda a derecha según el orden de la cláusula de Horn que usamos:*

1. $\neg M(x) \vee \neg D(x) \vee \neg CC(x, b)$
2. $\neg M(x) \vee \neg CC(x, y) \vee \neg C(y)$
3. $\neg D(x) \vee \neg CC(x, y) \vee \neg C(y)$

4. $\neg M(x)\neg D(x) \vee \neg C(y)$

5. $\neg D(x) \vee \neg CC(x,y) \vee \neg C(y)$

A partir de cada uno de estos nodos, que de nuevo son objetivos, se vuelven a abrir ramas al resolver con las cláusulas de Horn.

Tenemos que señalar en este momento que este proceso se puede convertir en automático. Sólo es necesario determinar el orden en que se examinan los objetivos parciales (cada uno de los literales que aparecen en el objetivo) y las cláusulas con un literal positivo.

9.3.3. Cómo surge un conjunto de Horn

Cabe preguntarse qué problemas de consecuencia lógica van a producir conjuntos de Horn cuando sean transformados a insatisfacibilidad de un conjunto de cláusulas. Podemos dar una sencilla respuesta sin más examinar los distintos tipos de cláusulas.

Un hecho representa a una afirmación sobre la veracidad de un predicado en un contexto: $C(b)$, es decir, C es cierto para la constante b, o $C(x)$ que recordemos que es la manera en la que escribimos $\forall x C(x)$, es decir, C es cierto para todo elemento del dominio.

Para aproximarnos al significado de una regla, digamos $\neg C(y) \vee \neg CC(x,y) \vee M(x)$ observamos que podemos, usando las leyes de Morgan, transformarla en la fórmula lógicamente equivalente $\neg(C(y) \wedge CC(x,y)) \vee M(x)$ que a su vez puede cambiarse por $(C(y) \wedge CC(x,y)) \rightarrow M(x)$. Ahora, recordemos que hay que incluir el cierre universal y leemos $\forall x \forall y[(C(y) \wedge CC(x,y)) \rightarrow M(x)]$: **Si** ocurre $C(y)$ y ocurre $CC(x,y)$, **entonces** ocurre $M(x)$ que tiene la forma del enunciado de una regla.

Por último fijemos nuestra atención en la cláusula negativa y pensemos que proviene de incluir la negación de la consecuencia en el conjunto de premisas; entonces el objetivo $\neg M(x)\neg D(x) \vee \neg CC(x,y) \vee \neg C(y)$ que, como antes, representa a su cierre universal, proviene de la negación de la fórmula

$$\exists x \exists y[M(x) \wedge D(x) \wedge CC(x,y) \wedge C(y)]$$

que es una pregunta de la forma ¿existen elementos para los que son ciertos simultáneamente todos los predicados?

Ejemplo 9.3.4. *Vamos a mostrar un ejemplo en el que trataremos de determinar si alguna frase es consecuencia lógica del siguiente conjunto de premisas:*

1. *Adán es una persona*
2. *Eva es una persona*
3. *Eva es la madre de Caín*
4. *Eva es la madre de Abel*
5. *Todo hijo de una persona es una persona*

Observamos que cada una de estas premisa está enunciada como un hecho o como una regla, en efecto, su traducción al lenguaje de primer orden con elementos

constantes: *e (Eva), c (Caín), a (Abel);*

Predicados: $P(x)$: *x es una persona;* $M(x,y)$: *x es la madre de y*

podría ser:

1. $P(a)$ *(hecho)*
2. $P(e)$ *(hecho)*
3. $M(e,c)$ *(hecho)*
4. $M(e,a)$ *(hecho)*

5. $\forall x \forall y(M(x,y) \wedge P(x) \rightarrow P(y)$ *(regla)*

y las correspondientes cláusulas (de Horn) que proporcionan (lo que se llamaría un programa en lenguaje PROLOG) son:

1. $P(a)$ *(hecho)*
2. $P(e)$ *(hecho)*
3. $M(e,c)$ *(hecho)*
4. $M(e,a)$ *(hecho)*
5. $P(y) \vee \neg M(x,y) \vee \neg P(x)$ *(regla)*

Este conjunto de premisas puede utilizarse para determinar si una serie de preguntas tienen respuesta afirmativa. Estas preguntas darán lugar al objetivo. Para este programa podemos formular preguntas como:

? *Es Eva una persona*

? *Es Caín una persona*

? *Hay alguna persona*

? *Es Caín hijo*

? *Tiene madre Abel*

? *Hay alguien que sea hijo de Eva.*

? *Tiene madre Eva*

que se traducen en el lenguaje de primer orden que estamos usando por las fórmulas:

1. $P(e)$,
2. $P(c)$,
3. $\exists x P(x)$,
4. $\exists x M(x,c)$,
5. $\exists x M(x,a)$
6. $\exists y M(e,y)$

Cada una de estas preguntas da lugar a un problema de consecuencia lógica y este se convierte en un problema de probar la insatisfacibilidad de un conjunto de Horn; además el conjunto usable es el mismo y sólo varía la cláusula objetivo de la que partimos.

9.3.4. Un programa y varios objetivos

Para el programa (es decir, el conjunto de cláusulas de Horn) del ejemplo anterior
y las correspondientes cláusulas (de Horn) que proporciona (lo que se llamaría un programa en lenguaje PROLOG) son:

1. $P(a)$ (hecho)
2. $P(e)$ (hecho)
3. $M(e,c)$ (hecho)
4. $M(e,a)$ (hecho)
5. $P(y) \vee \neg M(x,y) \vee \neg P(x)$ (regla)

desarrollaremos parte del árbol de las deducciones lineales-input correspondiente a algunos de los objetivos.

Ejemplo 9.3.5. *Queremos saber si puede deducirse de nuestro conjunto de premisas que Caín es una persona, es decir, si $P(c)$ es consecuencia lógica del conjunto de fórmulas inicial:*

$$\{P(a);\ P(e);\ M(e,c);\ M(e,a);\ \forall x\forall y(M(x,y)\wedge P(x)\to P(y)\}$$

En primer lugar tendremos que negar la conclusión :$\neg P(c)$ y a continuación calculamos las deduciones lineales-input partiendo del objetivo:

$$\begin{array}{c}\neg P(c)\\ \Big|\ {\scriptstyle 5,\,(y|c)}\\ \neg M(x,c)\vee\neg P(x)\\ \Big|\ {\scriptstyle 3,\,(x|e)}\\ \neg P(e)\\ \Big|\ {\scriptstyle 2}\\ \square\end{array}$$

Y puesto que hemos obtenido la cláusula vacía, la consecuecnia lógica ocurre.

Ejemplo 9.3.6. *Ahora nos preguntamos si hay alguien que sea hijo de Eva, es decir, queremos comprobar si $\exists y M(e,y)$ es consecuencia lógica de nuestro programa. Negando esta conclusión obtenemos la cláusula objetivo $\neg M(e,y)$ que es la raíz del árbol de las deducciones lineales-input:*

$$\begin{array}{c}\neg M(e,y)\\ \Big|\ {\scriptstyle 3,\,(y|c)}\\ \square\end{array}$$

y obtenemos que también es consecuencia lógica. Además, si seguimos el valor que toma la variable y para llegar a la cláusula vacía, obtenemos que $y=c$, lo cual se puede interpretar como una respuesta más concreta: Sí, hay alguien que es hijo de Eva y es Caín. Aunque hemos obtenido una respuesta satisfactoria, es obvio que hay otra respuesta posible que se deduce del conjunto de premisas ¿cómo puede obtenerse? Nos estamos adentrando en los procedimientos que se utilizan en el recorrido del árbol de las deducciones.

En la deducción anterior se calculó la resolvente del objetivo con la primera (en el orden dado) de las cláusulas de Horn y llegamos al nodo que contenía $\square$; podemos retroceder un paso y volvemos a estar een el objetivo $\neg M(e,y)$ que ahora tratamos de resolver con alguna cláusula que aparezca después de la ya utilizada: en efecto, también se puede resolver con la cláusula 4 $M(e,a)$ efectuando la sustitución $(y|a)$ y de nuevo llegamos a la cláusula vacía obteniendo la respuesta afirmativa: sí, hay alguien que es hijo de Eva y es Abel.

Ejemplo 9.3.7. *Por último veamos cuál es la respuesta a la pregunta ¿Tiene madre Eva? que se traduce por la fórmula $\exists x M(x,e)$ y que al ser negada da el objetivo $\neg M(x,e)$; pero en el conjunto usable no hay ninguna cláusula que conteniendo al predicado M como literal positivo sea unificable con $M(x,e)$. Así que como no es posible obtener la cláusula vacía mediante una deducción lineal-input, la respuesta es negativa: No, no podemos deducir que Eva tenga madre de las hipótesis consideradas.*