Notas de Clase para IL

3. Deducción en Lógica Proposicional
Ejercicios resueltos

Rafael Farré, Robert Nieuwenhuis,
Pilar Nivela, Albert Oliveras, Enric Rodríguez

3 de septiembre de 2009

1. Ejercicios

1. (dificultad 2) Demuestra que, para toda fórmula F, hay al menos una fórmula lógicamente equivalente que está en DNF. Ídem para CNF. Ayuda: obtener las fórmulas en CNF y DNF a partir de la tabla de verdad para F.

Dada una interpretación I, sea F_I la conjunción de todos los literales p_i tales que $I(p_i) = 1$, y de todos los literales $\neg p_i$ tales que $I(p_i) = 0$. Además, sea F' la disyunción de todas las F_I tales que $eval_I(F) = 1$. Por ejemplo, si F es $\neg((p \leftrightarrow q) \vee r)$, su tabla de verdad es:

p	q	r	$\neg((p \leftrightarrow q) \vee r)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

y la fórmula F' es $(\neg p \wedge q \wedge \neg r) \vee (p \wedge \neg q \wedge \neg r)$. Efectivamente, F' es una DNF, y además es lógicamente equivalente a F, puesto que para toda I se evalúa igual que F.

Similarmente, para obtener una CNF, basta con tomar la conjunción de las G_I tales que $eval_I(F) = 0$, donde cada G_I es la disyunción de todos los literales p_i tales que $I(p_i) = 0$, y de todos los literales $\neg p_i$ tales que $I(p_i) = 1$. ■

2. (dificultad 3) Da una manera de calcular una fórmula $\hat{F}$ en CNF para una fórmula F dada (con $\hat{F} \equiv F$) sin necesidad de construir previamente la tabla de verdad. Ayuda: aplica equivalencias lógicas como las leyes de De Morgan y la distributividad y el Lema de Sustitución.

El algoritmo va a consistir en dos fases. En la primera consideramos las siguientes tres equivalencias lógicas, expresadas como reglas de transformación de fórmulas:

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

Podemos aplicarlas sobre cualquier subfórmula. Por ejemplo, por la tercera regla, en $\neg(p \wedge \neg(q \vee \neg r))$ podemos reemplazar $\neg(q \vee \neg r)$, obteniendo $\neg(p \wedge (\neg q \wedge \neg\neg r))$, y después por la primera regla podemos obtener $\neg(p \wedge (\neg q \wedge r))$, etc.

Si aplicamos *exhaustivamente* (es decir, mientras sea posible) estas tres reglas, conseguiremos que todas las negaciones estén directamente aplicadas a los símbolos de predicado. Además, por el Lema de Sustitución, a cada aplicación de una regla, la fórmula permanece lógicamente equivalente.

En la segunda fase aplicamos exhaustivamente las reglas de distributividad

$$\begin{array}{lcl} (F \wedge G) \vee H & \Rightarrow & (F \vee H) \wedge (G \vee H) \\ F \vee (G \wedge H) & \Rightarrow & (F \vee G) \wedge (F \vee H) \end{array}$$

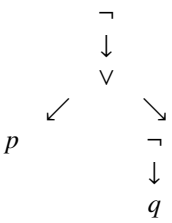
para obtener una CNF (nótese que, alternativamente, si se usan las reglas

$$\begin{array}{lcl} (F \vee G) \wedge H & \Rightarrow & (F \wedge H) \vee (G \wedge H) \\ F \wedge (G \vee H) & \Rightarrow & (F \wedge G) \vee (F \wedge H) \end{array}$$

entonces se obtiene una DNF). Nuevamente, por el Lema de Sustitución, a cada aplicación de las reglas la fórmula permanece lógicamente equivalente.

Veamos que el procedimiento que se acaba de describir proporciona un algoritmo, es decir, *termina*. Para ver que la aplicación exhaustiva de un conjunto de reglas termina, podemos definir una *medida* (un número natural) sobre fórmulas, y ver que esta medida decrece a cada paso de aplicación de una regla del conjunto.

Demostremos la terminación de la primera fase del procedimiento. Dada una fórmula F, su medida es la suma de, para cada símbolo proposicional y conectiva binaria ($\wedge$, $\vee$) que aparece, el número de $\neg$ que tiene por encima en su representación en árbol. Por ejemplo la medida de $\neg(p \vee \neg q)$, que vista como árbol es



es 1 + 1 + 2 = 4. Es fácil ver que las tres primeras reglas hacen decrecer esta medida, por lo que terminan.

Finalmente demostremos la terminación de la segunda fase del procedimiento. Para ello nos vamos a restringir sobre fórmulas en las que las negaciones se aplican directamente a símbolos de predicado: podemos hacer esta restricción porque las fórmulas cumplen esta propiedad después de la primera fase, y esta propiedad se preserva bajo las aplicaciones de las reglas de distributividad. Definimos la siguiente medida de manera recursiva:

$$\begin{array}{lcl} |F \wedge G| & = & |F| + |G| \\ |F \vee G| & = & |F|^2 \cdot |G|^2 \\ |p| & = & 1 \\ |\neg p| & = & 1 \end{array}$$

Nótese que la medida es siempre un número natural positivo. Vamos a demostrar que, si una aplicación de las reglas de distributividad transforma F en F', entonces $|F| > |F'|$. Lo haremos por inducción sobre la suma de $\wedge$ y $\vee$ en la fórmula F:

- **Caso base:** La suma de $\wedge$ y $\vee$ en F es cero, es decir, la fórmula sólo tiene negaciones y símbolos de predicado. En este caso el resultado se cumple trivialmente porque no se puede aplicar distributividad.
- **Paso de inducción:** La suma de $\wedge$ y $\vee$ en F es positiva, es decir, hay al menos una conectiva binaria en la fórmula. Como las negaciones sólo se aplican a símbolos de predicado, la fórmula F es o bien de la forma $G \wedge H$, o bien de la forma $G \vee H$.
  - Supongamos que F es de la forma $G \vee H$. Supongamos también que la regla se aplica a "nivel superior". Sin pérdida de generalidad podemos asumir que F es de la forma $(G_1 \wedge G_2) \vee H$, y que F' es entonces $(G_1 \vee H) \wedge (G_2 \vee H)$. Por lo tanto

    $$|F| = |G_1 \wedge G_2|^2 \cdot |H|^2 = (|G_1| + |G_2|)^2 \cdot |H|^2 = (|G_1|^2 + 2\,|G_1|\,|G_2| + |G_2|^2) \cdot |H|^2 >$$

    $$> |G_1|^2 \cdot |H|^2 + |G_2|^2 \cdot |H|^2 = |F'|$$

    Supongamos que la regla de distributividad se aplica a alguna de las subfórmulas G o H. Sin pérdida de generalidad, podemos suponer que se aplica sobre G: es decir, existe una fórmula G' obtenida mediante la aplicación de la regla de distributividad sobre G, que cumple que F' es $G' \vee H$. Luego $|F'| = |G'|^2 \cdot |H|^2$ y, por hipótesis de inducción, también se tiene que $|G| > |G'|$. Entonces $|F| = |G|^2 \cdot |H|^2 > |G'|^2 \cdot |H|^2 = |F'|$.

- Supongamos que F es de la forma $G \wedge H$. En este caso la regla de distributividad sólo se puede aplicar a alguna de las subfórmulas G o H, pero no a nivel superior. Sin pérdida de generalidad, podemos suponer que se aplica sobre G: es decir, existe una fórmula G' obtenida mediante la aplicación de la regla de distributividad sobre G, que cumple que F' es $G' \wedge H$. Luego $|F'| = |G'| + |H|$ y, por hipótesis de inducción, también se tiene que $|G| > |G'|$. Entonces $|F| = |G| + |H| > |G'| + |H| = |F'|$.

■

3. (dificultad 3) Cada fórmula de lógica proposicional puede verse como un circuito electrónico que tiene una puerta lógica por cada conectiva $\wedge$, $\vee$, $\neg$ que aparezca en la fórmula (aunque las fórmulas tienen estructura de árbol, mientras los circuitos en realidad permiten compartir subárboles repetidos, es decir, son grafos dirigidos acíclicos).

   El problema del *diseño lógico* consiste en encontrar un circuito adecuado que implemente una función booleana dada. Para conseguir circuitos *rápidos*, nos va bien representar la función booleana como una fórmula en CNF (o DNF), porque la *profundidad* del circuito será como máximo tres. Pero también es importante utilizar el mínimo número de conectivas (puertas lógicas). Los métodos de obtener CNFs vistos en los ejercicios anteriores, ¿nos dan la CNF más corta en ese sentido? ¿Se te ocurre alguna mejora?

4. (dificultad 1) La cláusula vacía $\square$ es el caso más sencillo de fórmula insatisfactible. Una CNF que es un conjunto de cero cláusulas, ¿es satisfactible o insatisfactible?

Esta CNF es la conjunción de cero fórmulas, y en la extensión de la lógica que hemos considerado, $eval_I(\bigwedge_{i \in 1..n} F_i) = min\{1, eval_I(F_1), \ldots, eval_I(F_n)\}$, lo cual da 1 si $n = 0$. Luego es una tautología. ■

5. (dificultad 2) Demuestra que una cláusula es una tautología si, y sólo si, contiene a la vez p y $\neg p$ para un cierto símbolo proposicional p.

- Por un lado, si $p \in C$ y $\neg p \in C$, entonces C es de la forma $p \vee \neg p \vee C'$ para una cierta cláusula C'; y como $(p \vee \neg p)$ es tautología, por la ley de tautología 2, C es una tautología.
- Demostremos el otro sentido por el *contrarecíproco* (demostrar $A \rightarrow B$ viendo que $\neg B \rightarrow \neg A$). Veamos que si para todo símbolo proposicional p se tiene $p \notin C$ o $\neg p \notin C$, entonces C no es una tautología. En efecto, definamos la interpretación I de la manera siguiente: para cada p, definamos $I(p) = 1$ si $p \notin C$, y si no $I(p) = 0$. Entonces para todo literal $l \in C$ se tiene que $eval_I(l) = 0$: si l es p, como $p \in C$, luego $eval_I(l) = I(p) = 0$; y si l es $\neg p$, entonces $\neg p \in C$, y necesariamente $p \notin C$, lo cual implica $eval_I(l) = 1 - I(p) = 0$. En consecuencia, $I \not\models C$, y por lo tanto C no es una tautología.

■

6. (dificultad 2) Sea S un conjunto de cláusulas con $\square \notin S$. Demuestra que S es satisfactible (dando un modelo para S) en cada una de las siguientes situaciones:

   a) Toda cláusula de S tiene algún literal positivo.

   b) Toda cláusula de S tiene algún literal negativo.

   c) Para todo símbolo de predicado p se cumple que: o bien p aparece sólo en literales positivos en S, o bien p aparece sólo en literales negativos en S.

---

a) Supongamos que toda cláusula de S tiene algún literal positivo. Definimos la interpretación I como $I(p) = 1$ para todo $p \in \mathcal{P}$. Veamos que $I \models C$ para toda cláusula C de S: en efecto, como por hipótesis existe $p \in \mathcal{P}$ tal que p es un literal (positivo) de C, tenemos que $eval_I(p) = 1$ y $eval_I(C) = 1$. Por lo tanto $I \models S$.

b) Supongamos que toda cláusula de S tiene algún literal negativo. Definimos la interpretación I como $I(p) = 0$ para todo $p \in \mathcal{P}$. Veamos que $I \models C$ para toda cláusula C de S: en efecto, como por hipótesis existe $p \in \mathcal{P}$ tal que $\neg p$ es un literal de C, tenemos que $eval_I(\neg p) = 1$ y $eval_I(C) = 1$. Por lo tanto $I \models S$.

c) Definimos una interpretación I de la manera siguiente: si p sólo aparece en literales positivos en S, definimos $I(p) = 1$; si p sólo aparece en literales negativos en S, definimos $I(p) = 0$. Claramente, la interpretación I está bien definida. Y es es un modelo de S: basta ver que $eval_I(C) = 1$ para toda cláusula C de S. Sea pues C una cláusula de S, que por hipótesis es no vacía. Distinguimos dos casos:

- Si C contiene un literal positivo p, entonces p sólo aparece en literales positivos en S, y por definición $I(p) = 1$. Luego $eval_I(C) = 1$.
- Si C contiene un literal negativo p, entonces p sólo aparece en literales negativos en S, y por definición $I(p) = 0$. Luego $eval_I(C) = 1$.

Como para cualquier cláusula C de S tenemos $eval_I(C) = 1$, I es un modelo de S y S es satisfactible.

■

7. (dificultad 2) Dados n símbolos proposicionales:

   a) ¿Cuántas cláusulas distintas (como conjuntos de literales) hay?

   b) ¿Cuántas de estas cláusulas son insatisfactibles?

   c) ¿Cuántas cláusulas distintas y que no son tautologías hay?

   d) ¿Cuántas cláusulas distintas que contienen exactamente un literal por cada símbolo proposicional hay?

8. (dificultad 4) Propón un algoritmo eficiente que, dado un conjunto de cláusulas S, retorna un conjunto de cláusulas S' (no necesariamente definido sobre los mismos símbolos de predicado que S) con como mucho 3 literales por cláusula, que es *equisatisfactible* a S (es decir, que es satisfactible si y sólo si S lo es). Ayuda: es posible introducir algún símbolo de predicado p nuevo, que signifique: "$l \vee l'$ es cierto" para algún par de literales l y l'.

Empezaremos motivando el algoritmo con un caso muy sencillo. Consideremos una CNF de una sola cláusula $l_1 \vee l_2 \vee l_3 \vee l_4$. Para conseguir un conjunto de cláusulas de como mucho 3 literales que sea equisatisfactible, consideraremos un símbolo proposicional adicional s cuyo significado será el de *"o bien l_3 o bien l_4 son ciertos"*. Mediante este símbolo adicional podemos obtener la fórmula equisatisfactible: $(l_1 \vee l_2 \vee s) \wedge (\neg s \vee l_3 \vee l_4)$.

Para el caso general, sea F la CNF $\bigwedge_{i=1}^{n} C_i$, donde cada C_i es una cláusula $\bigvee_{j=1}^{n_i} l^i_j$. A cada cláusula C_i le asociaremos el siguiente conjunto de cláusulas S_i:

$$
\begin{array}{lclcl}
l^i_1 & \vee & l^i_2 & \vee & s^i_1 \\
\neg s^i_1 & \vee & l^i_3 & \vee & s^i_2 \\
\neg s^i_2 & \vee & l^i_4 & \vee & s^i_3 \\
\dots & & & & \\
\neg s^i_{n_i-3} & \vee & l^i_{n_i-1} & \vee & l^i_{n_i}
\end{array}
$$

si C_i tiene más de 3 literales. En caso contrario S_i será simplemente C_i. Nótese que en S_i hemos introducido nuevos símbolos proposicionales s^i_j.

Vamos a ver intuitivamente por qué $\bigwedge_{i=1}^{n} S_i$ es equisatisfactible a F. Si I es un modelo de F, podemos construir un modelo I' para $\bigwedge_{i=1}^{n} S_i$ de la siguiente manera. I' coincidirá con I en los símbolos proposicionales existentes en F, y el valor de los símbolos s_j^i añadidos se definirá de la manera siguiente: $I(s_j^i) = 1$ si y sólo si alguno de los literales $\{l_{j+2}^i, \ldots, l_{n_i}^i\}$ es cierto en I. No es difícil ver que I' definida de esta manera es un modelo de $\bigwedge_{i=1}^{n} S_i$.

La otra implicación es más sencilla. Si tomamos un modelo I' de $\bigwedge_{i=1}^{n} S_i$, no es difícil ver que para cada cláusula C_i se cumple que alguno de los l_j^i es cierto en I'. Así pues, si restringimos I' a los símbolos proposicionales de F, obtendremos un modelo para F. ■

9. (dificultad 2) Sea S un conjunto de cláusulas de Horn con $\square \notin S$. Demuestra que S es satisfactible (dando un modelo para S) si no hay ninguna cláusula que sólo conste de un único literal positivo.

10. (dificultad 2) Demuestra que el enunciado del ejercicio previo es falso cuando S no es de Horn.

11. (dificultad 3) Definimos: un literal en una fórmula en CNF es *puro* si aparece o bien siempre negado o bien siempre sin negar. Además, se dice que una cláusula C es *redundante* si contiene al menos un literal puro. Demuestra que, si C' es una cláusula redundante, entonces $\{C_1, \ldots, C_n, C'\}$ es satisfactible si y sólo si $\{C_1, \ldots, C_n\}$ es satisfactible.

Si $C_1 \wedge \cdots \wedge C_n \wedge C'$ es satisfactible, el mismo modelo también satisface $\{C_1, \ldots, C_n\}$.

Veamos la implicación inversa. Supongamos que $\{C_1, \ldots, C_n\}$ es satisfactible. Entonces existe I tal que $I \models C_i$ para cada $i = 1 \ldots n$. Sea l un literal puro de C'. Distinguimos dos casos:

- Si $eval_I(l) = 1$, entonces $eval_I(C') = 1$, y $I \models \{C_1, \ldots, C_n, C'\}$.
- Si $eval_I(l) = 0$, sea I' la interpretación que es exactamente igual que I salvo para el símbolo de predicado de l, de manera que $eval_{I'}(l) = 1$. Veamos que $I' \models C_i$ para cada i en $1 \ldots n$. Como $I \models C_i$, entonces existe al menos un literal $l' \in C_i$ tal que $eval_I(l') = 1$. Como $eval_I(l) = 0$ y el literal l es puro, necesariamente l y l' no tienen el mismo símbolo proposicional. De modo que $eval_{I'}(l') = 1$ también, y $I' \models C_i$. Como por construcción $eval_{I'}(l) = 1$, entonces $eval_{I'}(C') = 1$, y por lo tanto $I' \models \{C_1, \ldots, C_n, C'\}$.

■

2. Ejercicios

12. (dificultad 3) Para una fórmula en DNF, ¿cuál es el mejor algoritmo posible para decidir si es satisfactible? ¿Qué coste tiene?

Una fórmula F en DNF es satisfactible si y sólo si lo es al menos una de sus conjunciones de literales (la demostración es similar a un ejercicio previo). Claramente una conjunción de literales es satisfactible si y sólo si no contiene un par de literales contradictorios p y $\neg p$ para cierto símbolo proposicional p. Así pues, nuestro algoritmo sólo debe decidir si se da esta situación o no. Esto se puede hacer en tiempo lineal de la siguiente manera.

Asumamos por simplicidad que los símbolos proposicionales de la fórmula son $p_1, \ldots, p_n$, donde n es menor que el tamaño de la entrada (si no es así, el algoritmo es más complicado al requerir *hashing*, pero sigue siendo lineal).

Nuestro algoritmo utilizará un vector v de n enteros que inicializamos a -1 (esto se puede hacer en tiempo lineal porque n es menor que el tamaño de la entrada). Para decidir si la primera conjunción tiene dos literales opuestos, la recorremos, y por cada literal l que vemos, si l es p_i ponemos $v[i]$ a 1, y si l es $\neg p_i$ ponemos $v[i]$ a 0. Al hacer esto podemos detectar si p_i ya ha sido visto con el signo opuesto

o no. Si al acabar de recorrer la conjunción no hemos visto ninguna pareja de opuestos, escribimos "sat" y acabamos. Si no, recorremos otra vez la misma conjunción para poner otra vez a -1 todas las $v[i]$s de sus literales y pasamos a la segunda conjunción. De esta manera procesamos todas las conjunciones. Al final, escribimos "insat" si hemos detectado parejas opuestas en todas ellas. ■

3. Ejercicios

13. (dificultad 2) Utiliza resolución para demostrar que $p \to q$ es una consecuencia lógica de

$$\begin{array}{rcl} t & \to & q \\ \neg r & \to & \neg s \\ p & \to & u \\ \neg t & \to & \neg r \\ u & \to & s \end{array}$$

14. (dificultad 2) Demuestra por resolución que son tautologías:

   a) $p \to (q \to p)$
   b) $(p \wedge (p \to q)) \to q$
   c) $((p \to q) \wedge \neg q) \to \neg p$
   d) $((p \to q) \wedge \neg q) \to \neg q$

Para ver que las fórmulas son tautologías veremos que sus negaciones son fórmulas insatisfactibles. Además, utilizaremos la equivalencia lógica $\neg(F \to G) \equiv (F \wedge \neg G)$, para fórmulas F y G arbitrarias.

a)

$$\begin{array}{rcl} \neg(p \to (q \to p)) & \equiv & p \wedge \neg(q \to p) \\ & \equiv & p \wedge (q \wedge \neg p) \\ & \equiv & p \wedge q \wedge \neg p \end{array}$$

Para demostrar insatisfactibilidad basta notar que de las cláusulas p y $\neg p$ se infiere $\square$.

b)

$$\begin{array}{rcl} \neg((p \wedge (p \to q)) \to q) & \equiv & p \wedge (p \to q) \wedge \neg q \\ & \equiv & p \wedge (\neg p \vee q) \wedge \neg q \end{array}$$

A partir del conjunto de cláusulas $\{p, \neg p \vee q, \neg q\}$ obtenemos la cláusula vacía mediante:

$$\dfrac{\dfrac{\neg p \vee q \quad \neg q}{\neg p} \qquad p}{\square}$$

c)

$$\begin{array}{rl} \neg(((p \to q) \wedge \neg q) \to \neg p) & \equiv ((p \to q) \wedge \neg q) \wedge p \\ & \equiv (\neg p \vee q) \wedge \neg q \wedge p \end{array}$$

Obtenemos la cláusula vacía utilizando las mismas resoluciones que en el apartado anterior.

d)

$$\begin{array}{rl} \neg(((p \to q) \wedge \neg q) \to \neg q) & \equiv ((p \to q) \wedge \neg q) \wedge q \\ & \equiv (\neg p \vee q) \wedge \neg q \wedge q \end{array}$$

Para demostrar insatisfactibilidad basta notar que de las cláusulas q y $\neg q$ se infiere $\square$.

■

15. (dificultad 2) Demuestra que la resolución es correcta.

En primer lugar, vamos a demostrar que dadas dos cláusulas $p \vee C$ y $\neg p \vee D$, entonces $(p \vee C) \wedge (\neg p \vee D) \models C \vee D$. Tenemos que ver que todo modelo I de $(p \vee C) \wedge (\neg p \vee D)$ también satisface $C \vee D$. Distinguimos dos casos:

- si $I(p) = 1$, entonces $eval_I((p \vee C) \wedge (\neg p \vee D)) = 1$ implica $eval_I(\neg p \vee D) = 1$, luego $eval_I(D) = 1$.
- si $I(p) = 0$, entonces $eval_I((p \vee C) \wedge (\neg p \vee D)) = 1$ implica $eval_I(p \vee C) = 1$, luego $eval_I(C) = 1$.

En cualquier caso, $eval_I(C \vee D) = 1$. Así, $(p \vee C) \wedge (\neg p \vee D) \models C \vee D$.

Vamos a demostrar ahora el enunciado del ejercicio. Por definición de corrección, tenemos que ver que, dado un conjunto de cláusulas S, para toda cláusula C tal que $C \in Res(S)$, entonces $S \models C$. Por otra parte, por definición $Res(S) = \cup_{i=0}^{\infty} S_i$, donde $S_0 = S$ y $S_i = S_{i-1} \cup \{ C \vee D \mid p \vee C \in S_{i-1}, \neg p \vee D \in S_{i-1}\}$ $(i > 0)$. Basta pues ver que para todo $i \geq 0$ y para toda cláusula C tal que $C \in S_i$, entonces $S \models C$.

Veamos esto último por inducción sobre i. Para $i = 0$, como $S_0 = S$, entonces para toda cláusula C tal que $C \in S_0 = S$ se tiene $S \models C$ trivialmente. Supongamos pues que $i > 0$. Entonces, dada una cláusula C de S_i, tenemos dos casos:

- $C \in S_{i-1}$. Entonces, por hipótesis de inducción, $S \models C$.
- $C \notin S_{i-1}$. Entonces C es de la forma $C' \vee D'$, donde $p \vee C', \neg p \vee D' \in S_{i-1}$. Por lo que hemos demostrado previamente, $(p \vee C') \wedge (\neg p \vee D') \models C' \vee D'$. Por hipótesis de inducción, tenemos $S \models p \vee C'$ y $S \models \neg p \vee D'$, de donde $S \models (p \vee C') \wedge (\neg p \vee D')$. Por la transitividad de la consecuencia lógica, de ahí finalmente se deduce que $S \models C' \vee D'$, esto es, $S \models C$.

■

16. (dificultad 2) Demuestra que, para todo conjunto finito de cláusulas S, $Res(S)$ es un conjunto finito de cláusulas, si se consideran las cláusulas como conjuntos de literales (por ejemplo, $C \vee p$ es la misma cláusula que $C \vee p \vee p$).

Es fácil ver que la regla de resolución no introduce símbolos proposicionales nuevos. Así pues, cualquier cláusula de $Res(S)$ contendrá sólo símbolos proposicionales que ya aparecen en S. Finalmente es trivial ver que sólo hay un número finito de cláusulas (si se consideran como conjuntos de literales) que utilicen los n símbolos proposicionales de S. Más concretamente, existen 2^{2n} cláusulas distintas (ver ejercicio previo). ■

17. (dificultad 3) Sea S un conjunto de cláusulas. Demuestra que $Res(S)$ es lógicamente equivalente a S.

Tenemos que ver que cualquier modelo de S es un modelo de $Res(S)$ y viceversa. Por un lado, cualquier modelo de $Res(S)$ es un modelo de S ya que todas las cláusulas de S están en $Res(S)$.

Por otro lado, veamos que cualquier modelo de S es un modelo de $Res(S)$. Sea I una interpretación tal que $I \models S$. Como la resolución es correcta, para toda cláusula $C \in Res(S)$ tenemos $S \models C$. Luego $I \models C$, y como C es arbitraria, $I \models Res(S)$.

■

18. (dificultad 2) ¿La resolución es completa? Demuéstralo.

La resolución no es completa. Por ejemplo, $p \models p \vee q$, pero no podemos obtener $p \vee q$ a partir de p mediante resolución. ■

19. (dificultad 2) Sea S un conjunto de cláusulas insatisfactible. Por la completitud refutacional de la resolución, sabemos que existe una demostración por resolución de que $\square \in Res(S)$. ¿Es esta demostración única?

20. (dificultad 4) Demuestra la completitud refutacional de la resolución, esto es, si S es un conjunto de cláusulas insatisfactible entonces $\square \in Res(S)$.

    Ayuda: demuestra el contrarrecíproco por inducción sobre el número N de símbolos de predicado de S.

Demostramos que si $\square \notin Res(S)$ entonces S es satisfactible, por inducción sobre N.

- $N = 0$. Entonces S sólo contiene la cláusula vacía (que no es el caso porque tendríamos $\square \in Res(S)$) o no contiene ninguna cláusula (y entonces es trivialmente satisfactible).
- $N > 0$. Sea p un símbolo de predicado de S.
  Sea SP (Sin P) el conjunto de cláusulas de S que no contienen el símbolo p.
  Sea $\{\ p \vee C_1,\ \ldots,\ p \vee C_n,\ \neg p \vee D_1,\ \ldots,\ \neg p \vee D_m\ \}$ el conjunto de todas las cláusulas no-tautológicas de S con el símbolo p.
  Sea $S' = SP\ \cup\ \{\ C_i \vee D_j \mid i \in 1..n,\quad j \in 1..m\ \}$, que es un conjunto en el que p no aparece; tiene $N - 1$ símbolos de predicado.
  Puesto que todas las cláusulas de S' están en $Res(S)$, si $\square \notin Res(S)$ entonces $\square \notin Res(S')$. Esto implica por H.I. que S' es satisfactible, es decir, tiene un modelo I'. Ahora veremos que esto implica que S es satisfactible, construyendo un modelo I para S, *extendiendo* el modelo I', que no está definido para p al no aparecer p en S'. Hay dos casos:
  - $I' \models C_i$ para toda $i \in 1..n$.
    Entonces la extensión I de I' con $I(p) = 0$ es modelo de S: satisface todas las cláusulas (no-tautológicas) de S que no están en S', que son las del conjunto
    $\{\ p \vee C_1,\ \ldots,\ p \vee C_n,\ \neg p \vee D_1,\ \ldots,\ \neg p \vee D_m\ \}$.
  - $I' \not\models C_k$ para alguna $k \in 1..n$.
    Entonces, como S' contiene $\{\ C_k \vee D_1,\ \ldots,\ C_k \vee D_m\ \}$ y $I' \models S'$, tenemos que $I' \models D_j$, para toda $j \in 1..m$. Entonces, como antes, la extensión I de I' con $I(p) = 1$ es modelo de S.

■

21. (dificultad 2) Demuestra que el lenguaje de las cláusulas de Horn es cerrado bajo resolución, es decir, a partir de cláusulas de Horn por resolución sólo se obtienen cláusulas de Horn.

22. (dificultad 2) Considera el siguiente caso particular de la resolución:

$$\frac{p \qquad \neg p \vee C}{C} \qquad \textit{Resolución Unitaria}$$

    Demuestra que la resolución unitaria es correcta.

23. (dificultad 2) Demuestra que la resolución unitaria no es refutacionalmente completa para cláusulas que no son de Horn.

El conjunto

$$\begin{array}{ll} p \vee q & \neg p \vee q \\ p \vee \neg q & \neg p \vee \neg q \end{array}$$

es un conjunto de cláusulas insatisfactible, ya que existe la siguiente refutación utilizando resolución:

$$\dfrac{\dfrac{p \vee q \quad p \vee \neg q}{p} \qquad \dfrac{\neg p \vee q \quad \neg p \vee \neg q}{\neg p}}{\square}$$

pero resolución unitaria no puede detectar su insatisfactibilidad porque no es aplicable. ■

24. (dificultad 3) Demuestra que la resolución unitaria es refutacionalmente completa para cláusulas de Horn. Ayuda: Basta con ver que, si S es un conjunto de cláusulas de Horn y $\square \notin ResUnit(S)$, entonces $ResUnit(S)$ (y por lo tanto S) tiene un modelo I. Define I como $I(p) = 1$ si y sólo si p es una cláusula (de un solo literal) en $ResUnit(S)$ y demuestra $I \models ResUnit(S)$ por inducción sobre el número de literales de las cláusulas.

Sea S un conjunto de cláusulas de Horn. Tenemos que demostrar que si S es insatisfactible, entonces $\square \in ResUnit(S)$. Por el contrarecíproco, basta con ver que, si $\square \notin ResUnit(S)$, entonces $ResUnit(S)$ (y por lo tanto S, puesto que $ResUnit(S) \supseteq S$) tiene un modelo I, es decir, es satisfactible. Definimos I como $I(p) = 1$ si y sólo si p es una cláusula (de un solo literal) en $ResUnit(S)$.

Ahora demostremos que $I \models ResUnit(S)$. Para ello basta con ver que $I \models C$ para toda cláusula C de $ResUnit(S)$. Lo hacemos por inducción sobre $|C|$, el número de literales de C:

- Caso base: $|C| = 0$. Claramente $I \models C$ para toda cláusula C de $Res(S)$ de cero literales, puesto que $\square \notin ResUnit(S)$.
- Paso de inducción: $|C| > 0$. Si C es una cláusula de un solo literal positivo p, tenemos que $I \models C$ por definición de I. En caso contrario, tratándose de cláusulas de Horn, C necesariamente debe contener algún literal negativo, es decir, C es de la forma $\neg p \vee C'$. Si $I(p) = 0$, tenemos $I \models C$. Si $I(p) = 1$, eso es debido a que existe una cláusula unitaria p en $ResUnit(S)$ (por la manera en que hemos definido I). Pero entonces, como tenemos $\neg p \vee C' \in ResUnit(S)$ y $p \in ResUnit(S)$, tenemos $C' \in ResUnit(S)$. Como $|C'| = |C| - 1$, por hipótesis de inducción tenemos $I \models C'$, lo cual implica $I \models C$.

■

25. (dificultad 2) ¿Cuál es la complejidad del problema de determinar si un conjunto de cláusulas de Horn S es satisfactible? Ayuda: analiza (informalmente) la corrección y la complejidad del siguiente algoritmo (que intenta construir sistemáticamente el modelo *minimal* I de S):

(0) inicialmente, $I(p) = 0$ para todo p
(1) hacer ciertos en I los p que son cláusulas unitarias positivas;
(2) eliminar de todas las cláusulas los literales $\neg p$ con $I(p) = 1$;
    si esto da lugar a la cláusula vacía: insatisfactible
    si no, si esto da lugar a alguna cláusula unitaria nueva, volver a (1)
        si no, la interpretación I construida es un modelo.

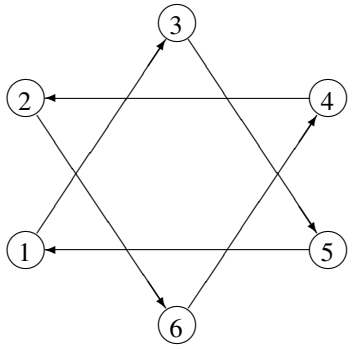
Para demostrar la corrección, veamos primero que el algoritmo devolverá insatisfactible sólo cuando S realmente lo sea. El algoritmo de insatisfactible sólo si se genera la cláusula vacía. Pero es también claro que todas las cláusulas generadas por el algoritmo son de hecho inferencias por resolución unitaria. Así pues, debido a la corrección de la resolución, si se puede inferir la cláusula vacía a partir de S necesariamente S es insatisfactible.

Veamos ahora que si el algoritmo nos devuelve I, entonces I es de hecho un modelo para S. El algoritmo terminará con un cierto conjunto de cláusulas S' tal que para toda cláusula C de S existe otra cláusula C' en S' con $C' \subseteq C$ (viendo las cláusulas como el conjunto de sus literales), ya que sólo eliminamos literales de cláusulas de S. Así pues, si demostramos que $I \models S'$ entonces también $I \models S$. Para ello será importante notar que en las cláusulas de S' no aparece ningún literal $\neg p$ con $I(p) = 1$, ya que los eliminamos en (2). Sea D una cláusula de S' (sabemos que D no será la cláusula vacía), y veamos que tiene que ser cierta en I. Si D contiene algún literal negativo $\neg p$ entonces D es cierta en I, ya que $I(p) = 0$. En caso contrario sólo puede ser de la forma p, y entonces en el paso (1) del algoritmo habremos hecho p cierto en I.

En cuanto a la complejidad, vemos que sólo se hace trabajo cada vez que se añade una nueva cláusula unitaria positiva, de las cuales sólo se pueden generar como mucho n (si n es el número de símbolos proposicionales). Dada una cláusula unitaria positiva p, el trabajo consiste en recorrer la fórmula eliminando todas las apariciones de $\neg p$, trabajo lineal en el tamaño de la fórmula. Así pues, el trabajo a hacer es como mucho $n \cdot m$ si m es el tamaño de la fórmula, que en el caso peor es cuadrático en el tamaño de la entrada. Nota adicional: con estructuras de datos adecuadas (en las que aquí no entraremos), este algoritmo de hecho es lineal. ■

26. (dificultad 2) Las *cláusulas de Krom* son aquellas que tienen a lo sumo dos literales. ¿Cuántas cláusulas de Krom se pueden construir con n símbolos de predicado? Demuestra que basta un número cuadrático de pasos de resolución para decidir si un conjunto de cláusulas de Krom es satisfactible o no.

27. (dificultad 3) Un *grafo* $G = (V, E)$ es un conjunto V de objetos llamados *vértices* conectados por enlaces llamados *aristas*; la arista que une los vértices u y v se representa como el par de vértices (u, v), y el conjunto de aristas se denota por E. Además, se dice que un grafo está *dirigido* si se distingue entre las dos posibles orientaciones de las aristas. Por ejemplo, en el grafo dirigido siguiente:



el conjunto de vértices V es $\{1, 2, 3, 4, 5, 6\}$, y el de aristas E es $\{(1, 3), (3, 5), (5, 1), (2, 6), (6, 4), (4, 2)\}$. Finalmente, se dice que una secuencia de vértices v_0, ..., v_k es un *camino* si se tiene que los vértices v_0, ..., v_k están sucesivamente conectados, es decir, si $(v_{i-1}, v_i) \in E$ para todo $1 \leq i \leq k$. Por ejemplo, 1, 3 y 5 forman un camino del grafo dibujado más arriba, ya que $(1, 3) \in E$ y $(3, 5) \in E$. Los grafos son objetos muy importantes en matemáticas e informática y se estudian en detalle en varias asignaturas posteriores.

Dado S un conjunto de cláusulas con 1 ó 2 literales por cláusula, definido sobre los símbolos proposicionales p_1, ..., p_n, se define el *grafo asociado a* S, denotado G_S, como el grafo dirigido $G_S = (V, E)$, donde $V = \{p_1, \ldots, p_n, \neg p_1, \ldots, \neg p_n\}$ y $E = \{(l, l') \mid \neg l \vee l' \in S\}$ (en este ejercicio, abusando de la notación, dado un literal l de la forma $\neg p$, vamos a considerar que $\neg l$ representa p; además, en la construcción del grafo las cláusulas de 1 literal, o sea de la forma p, se consideran como $p \vee p$).

a) Demuestra que si hay un camino de l a l' en G_S, entonces $\neg l \vee l' \in Res(S)$. Recíprocamente, demuestra que si $l \vee l' \in Res(S)$, entonces hay un camino de $\neg l$ a l' en G_S.

b) Demuestra que S es insatisfactible si y sólo si existe un símbolo proposicional p tal que hay un camino en G_S de p a $\neg p$, y otro camino de $\neg p$ a p.

c) Basándote en el apartado previo, propón un algoritmo para determinar la satisfactibilidad de un conjunto de cláusulas con 1 ó 2 literales por cláusula. ¿Qué complejidad tiene, en términos del número de cláusulas y de símbolos proposicionales de S?

---

a) Veamos que si hay un camino de l a l' en G_S, entonces $\neg l \vee l' \in Res(S)$. Sean l y l' literales tales que existe un camino $l_0 \Longrightarrow \cdots \Longrightarrow l_k$ en G_S, donde l es l_0 y l' es l_k. Esto implica la existencia de cláusulas en S de la forma $\neg l_0 \vee l_1$, $\neg l_1 \vee l_2, \ldots, \neg l_{k-1} \vee l_k$. Aplicando sucesivos pasos de resolución, podemos obtener la cláusula $\neg l_0 \vee l_k$, de modo que $\neg l \vee l' \in Res(S)$.

Recíprocamente, veamos que si $l \vee l' \in Res(S)$, entonces hay un camino de $\neg l$ a l' en G_S. Vamos a demostrarlo por inducción sobre N, el número de pasos de resolución necesarios para obtener $l \vee l'$ a partir de S.

Para $N = 0$, entonces $l \vee l' \in S$, y por construcción de G_S, existe una arista $(\neg l, l') \in E$. Para $N > 0$, sea p el símbolo proposicional sobre el que se ha resuelto en el último paso de resolución aplicado para obtener $l \vee l'$ a partir de S. Distinguimos varios casos según como sean las premisas usadas en esa última resolución:

- Si las premisas son de la forma $l \vee p$, $\neg p \vee l'$, entonces por hipótesis de inducción existen caminos de $\neg l$ a p y de p a l'. Luego hay un camino de $\neg l$ a l'.
- Si las premisas son de la forma p, $\neg p \vee l'$, entonces l es l'. Por hipótesis de inducción, existen caminos de $\neg p$ a p, de p a l', y de $\neg l'$ a $\neg p$. Luego hay un camino de $\neg l$ a l'.
- Si las premisas son de la forma $p \vee l$, $\neg p$, entonces l es l'. Por hipótesis de inducción, existen caminos de p a $\neg p$, de $\neg l$ a p, y de $\neg p$ a l'. Luego hay un camino de $\neg l$ a l'.

b) Por la corrección y la completitud refutacional de la resolución, S es insatisfactible si y sólo si $\square \in S$. Esto ocurre si y sólo si existe un símbolo proposicional p tal que $p, \neg p \in Res(S)$. Y por el apartado anterior, eso ocurre si y sólo si hay caminos en G_S de $\neg p$ a p, y de p a $\neg p$.

c) Podemos pues, construir el grafo G_S y para cada símbolo proposicional p, mirar si hay un camino de p a $\neg p$ y de $\neg p$ a p.

Representamos vértices con enteros y el grafo con *listas de adyacencia*: guardamos por cada vértice v_i una lista (por ej. un vector) con los vértices v_j tales que $(v_i, v_j) \in E$. Para ver si existe un camino de un vértice u a otro v podemos calcular los vértices *alcanzables* desde u: son los vértices en la lista de adyacencia de u, y los alcanzables desde éstos (marcamos los vértices ya visitados para no repetir trabajo). Esto tiene coste lineal en la suma de los vértices y de las aristas. Si tenemos n símbolos proposicionales y m cláusulas, el coste del algoritmo es proporcional a $n(n + m)$.

De hecho, este algoritmo no es óptimo: aquí no lo veremos en detalle, pero hay un algoritmo lineal que divide V en sus *componentes fuertemente conexas* (conjuntos de vértices $\{v_1, \ldots, v_k\}$ tales que hay camino de todo v_i a todo v_j y viceversa) y comprueba si alguna componente contiene algún par de la forma p y $\neg p$.

■

4. Ejercicios

28. (dificultad 3) Dado un mapa de un continente, es posible colorearlo con cuatro colores sin que dos países con frontera común tengan el mismo color (es el famoso *four color problem*). Para grafos, el problema se generaliza al problema NP-completo de *K-coloreado:* dado un grafo G y un número

natural K, decidir si podemos asignar a cada vértice un natural entre 1 y K (un *color*), tal que todo par de vértices adyacentes tengan colores distintos.

Expresa este problema mediante una CNF, de modo que se pueda resolver con un SAT solver. ¿Cuántos símbolos de predicado se necesitan? ¿Cuántas cláusulas se obtienen?

Codificamos el problema del K-coloreado del grafo $G = (V, E)$ en SAT de la siguiente manera. Definimos, por cada vértice $v \in V$ y por cada color k ($1 \leq k \leq K$) el símbolo proposicional $p_{v,k}$, con el significado "el vértice v está pintado con color k"; con ello se obtienen $|V| \cdot K$ símbolos de predicado. Entonces el problema se reduce a la satisfactibilidad del siguiente conjunto de cláusulas, que codifican:

- **Cada vértice tiene al menos un color.** Para eso, incluimos por cada vértice $v \in V$ una cláusula de la forma $p_{v,1} \vee \ldots \vee p_{v,K}$. En total, son $|V|$ cláusulas de K literales. Nota: esto es muy similar al caso 1 del ejemplo de los sudokus ("cada casilla tiene al menos un valor").
- **Cada vértice no tiene más de un color.** Para eso, incluimos una cláusula binaria por cada vértice $v \in V$ y por cada par de colores k_1 y k_2 (con $1 \leq k_1 < k_2 \leq K$): $\neg p_{v,k_1} \vee \neg p_{v,k_2}$. Nótese que la condición $k_1 < k_2$ es necesaria ya que los colores deben ser distintos. También nos ahorran la repetición de cláusulas: $\neg p_{v,1} \vee \neg p_{v,2}$ es lógicamente equivalente a $\neg p_{v,2} \vee \neg p_{v,1}$, pero la condición $k_1 < k_2$ sólo nos permite expresar una de ellas. En total, son $|V| \cdot \binom{K}{2}$ cláusulas. Nota: esto es muy similar al caso 2 del ejemplo de los sudokus ("en cada casilla no hay más de un valor").
- **Todo par de vértices adyacentes tiene colores distintos.** Para eso, incluimos una cláusula binaria por cada par de vértices $v_1, v_2 \in V$ tales que $(v_1, v_2) \in E$ y por cada color k (con $1 \leq k \leq K$): $\neg p_{v_1,k} \vee \neg p_{v_2,k}$. En total, son $|E| \cdot K$ cláusulas. Nota: esto es muy similar al caso 3 del ejemplo de los sudokus.

■

29. (dificultad 3) Dados un edificio de una sola planta con muchos pasillos rectos que se cruzan, y un número natural K, ¿se pueden colocar cámaras giratorias en los cruces de los pasillos de modo que sea posible vigilar todos los pasillos con como mucho K cámaras? Este problema se puede formalizar como el problema de grafos llamado *vertex cover* o, traducido, *recubrimiento de vértices*: ¿existe un subconjunto de tamaño como mucho K de los N vértices, el *recubrimiento*, tal que toda arista tenga al menos un extremo en el recubrimiento (es decir, quede *cubierta*)?

    Expresa este problema mediante una CNF, de modo que se pueda resolver con un SAT solver. Usa los $K \cdot N$ símbolos de predicado $p_{i,j}$ que signifiquen: "el i-ésimo miembro del recubrimiento (con i entre 1 y K) es el vértice j (con j entre 1 y N)". ¿Cuántas cláusulas se obtienen?

30. (dificultad 5) Siguiendo el problema anterior, si hubiésemos usado la codificación con símbolos de predicado p_i que significasen: "hay una cámara en el vértice i", ¿cómo expresarías de forma compacta que no hay más de K cámaras? (Ayuda: piensa en circuitos sumadores y usa símbolos adicionales). Usando estos símbolos de predicado expresa el problema de *vertex cover* mediante una CNF, de modo que se pueda resolver con un SAT solver.

31. (dificultad 3) Dado un grupo de estudiantes con las listas de asignaturas que estudia cada uno, y un natural K, ¿hay algún subconjunto de exactamente K estudiantes tal que toda asignatura tenga algún estudiante que la curse?

    Expresa este problema mediante una CNF, de modo que se pueda resolver con un SAT solver. ¿Cuántos símbolos de predicado se necesitan? ¿Cuántas cláusulas se obtienen?

Codificamos el problema en SAT de la siguiente manera. Sean $E = \{e_1, e_2, ..., e_N\}$ el conjunto de estudiantes y A el conjunto de asignaturas. Definimos, por cada k tal que $1 \leq k \leq K$ y por cada $1 \leq n \leq N$, el símbolo proposicional $p_{k,n}$, con el significado "el k-ésimo estudiante es e_n"; con ello se obtienen $K \cdot N$ símbolos proposicionales. Entonces el problema se reduce a la satisfactibilidad del siguiente conjunto de cláusulas, que codifican:

- **El k-ésimo estudiante es un estudiante.** Para eso, incluimos una cláusula por cada k tal que $1 \leq k \leq K$ de la forma $p_{k,1} \vee p_{k,2} \vee \cdots \vee p_{k,N}$. En total, son K cláusulas.
- **El k-ésimo estudiante sólo puede ser uno.** Para eso, incluimos una cláusula binaria por cada k tal que $1 \leq k \leq K$ y por cada par de estudiantes distintos, es decir, para cada n_1, n_2 tales que $1 \leq n_1 < n_2 \leq N$: $\neg p_{k,n_1} \vee \neg p_{k,n_2}$. En total, son $K \cdot \binom{N}{2}$ cláusulas.
- **Los estudiantes del subconjunto son distintos entre sí.** Para eso, incluimos una cláusula binaria por cada par de índices k_1, k_2 tales que $1 \leq k_1 < k_2 \leq K$ y por cada estudiante, es decir, por cada $1 \leq n \leq N$: $\neg p_{k_1,n} \vee \neg p_{k_2,n}$. En total, son $N \cdot \binom{K}{2}$ cláusulas.
- **Toda asignatura tiene algún estudiante que la cursa.** Para eso, incluimos una cláusula por cada asignatura $a \in A$ expresando que al menos uno de los estudiantes que la cursan es uno de los K estudiantes del subconjunto. Si llamamos e_{n_1}, e_{n_2}, ... e_{n_p} los estudiantes que cursan la asignatura a, la cláusula es

  $$\begin{array}{llllll} (& p_{1,n_1} \vee p_{2,n_1} & \vee & \cdots & \vee p_{K,n_1}) & \vee \\ (& p_{1,n_2} \vee p_{2,n_2} & \vee & \cdots & \vee p_{K,n_2}) & \vee \\ & \cdots & & & & \vee \\ (& p_{1,n_p} \vee p_{2,n_p} & \vee & \cdots & \vee p_{K,n_p}) & \end{array}$$

  En total, son $|A|$ cláusulas.

En conclusión, se requieren un total de $K + K \cdot \binom{N}{2} + N \cdot \binom{K}{2} + |A|$ cláusulas.

■

32. (dificultad 3) ¿Cuál crees que es el coste mínimo de un algoritmo que calcule una DNF lógicamente equivalente para una fórmula F?

En un ejercicio previo se ha visto que, para fórmulas en DNF, el problema de satisfactibilidad se puede resolver en coste lineal. Ahora bien: si el coste de calcular la DNF de una fórmula cualquiera fuese polinómico, esto implicaría que entonces también lo sería determinar la satisfactibilidad de fórmulas cualesquiera, lo cual permitiría resolver todos los problemas NP-completos en tiempo polinómico. Luego no parece probable que exista ningun algoritmo que calcule la DNF de una fórmula dada en tiempo polinómico. De hecho, se sabe que hay fórmulas que no tienen ninguna DNF equivalente cuyo tamaño no sea exponencialmente mayor que el tamaño de F (y calcular algo exponencialmente grande y escribirlo obviamente requiere tiempo exponencial). ■

5. Ejercicios

33. (dificultad 2) Di cuáles de las siguientes fórmulas son satisfactibles utilizando el procedimiento DPLL:

   a) $(p \vee \neg q \vee r \vee \neg s) \wedge (\neg r \vee s) \wedge q \wedge \neg p$

   b) $(p \vee q) \wedge (\neg p \vee \neg q) \wedge (p \vee r) \wedge (\neg p \vee \neg r)$

   c) $(p \vee q) \wedge (\neg p \vee \neg q) \wedge (p \vee r) \wedge (\neg p \vee \neg r) \wedge (q \vee r) \wedge (\neg q \vee \neg r)$

En este ejercicio y en los que siguen, vamos a denotar la negación de un símbolo proposicional con una rayita; por ejemplo, $\bar{q}$ denota $\neg q$. Además, anotamos cada $\Longrightarrow$ con la primera letra de la regla aplicada (Propaga, Decide, Falla, o Backtrack), y el número de la cláusula usada cuando sea necesario.

a) El conjunto de cláusulas del cual queremos estudiar la satisfactibilidad es

$$\begin{array}{lllllllll} 1. & p & \vee & \bar{q} & \vee & r & \vee & \bar{s} \\ 2. & & & & & \bar{r} & \vee & s \\ 3. & & & q & & & & \\ 4. & \bar{p} & & & & & & \end{array}$$

Entonces tenemos: $\emptyset \Longrightarrow_{p3} q \Longrightarrow_{p4} q\,\bar{p} \Longrightarrow_{d} q\,\bar{p}\,r^{\mathsf{d}} \Longrightarrow_{p2} q\,\bar{p}\,r^{\mathsf{d}}\,s$

Como ya no podemos aplicar ninguna otra regla, la interpretación I tal que $I(p) = 0$, $I(q) = 1$, $I(r) = 1$, $I(s) = 1$, es un modelo del conjunto de cláusulas, que es por lo tanto satisfactible.

b) El conjunto de cláusulas del cual queremos estudiar la satisfactibilidad es

$$\begin{array}{lllllll} 1. & p & \vee & q & & & \\ 2. & \bar{p} & \vee & \bar{q} & & & \\ 3. & p & & & \vee & r \\ 4. & \bar{p} & & & \vee & \bar{r} \end{array}$$

Entonces tenemos: $\emptyset \Longrightarrow_{d} p^{\mathsf{d}} \Longrightarrow_{p2} p^{\mathsf{d}}\,\bar{q} \Longrightarrow_{p4} p^{\mathsf{d}}\,\bar{q}\,\bar{r}$

Como ya no podemos aplicar ninguna otra regla, la interpretación I tal que $I(p) = 1$, $I(q) = 0$, $I(r) = 0$, es un modelo del conjunto de cláusulas, que es por lo tanto satisfactible.

c) El conjunto de cláusulas del cual queremos estudiar la satisfactibilidad es

$$\begin{array}{lllllll} 1. & p & \vee & q & & & \\ 2. & \bar{p} & \vee & \bar{q} & & & \\ 3. & p & & & \vee & r \\ 4. & \bar{p} & & & \vee & \bar{r} \\ 5. & & & q & \vee & r \\ 6. & & & \bar{q} & \vee & \bar{r} \end{array}$$

Entonces tenemos: $\emptyset \Longrightarrow_{d} p^{\mathsf{d}} \Longrightarrow_{p2} p^{\mathsf{d}}\,\bar{q} \Longrightarrow_{p4} p^{\mathsf{d}}\,\bar{q}\,\bar{r} \Longrightarrow_{b5} \bar{p} \Longrightarrow_{p1} \bar{p}\,q \Longrightarrow_{p3} \bar{p}\,q\,r \Longrightarrow_{f6}$ "*Insat*"

Como finalmente hemos terminado con el estado "*Insat*", el conjunto de cláusulas es insatisfactible.

■

34. (dificultad 2) Utiliza el procedimiento DPLL para demostrar que $p \rightarrow q$ es una consecuencia lógica de

$$\begin{array}{rcl} t & \rightarrow & q \\ \neg r & \rightarrow & \neg s \\ p & \rightarrow & u \\ \neg t & \rightarrow & \neg r \\ u & \rightarrow & s \end{array}$$

35. (dificultad 2) Demuestra que son tautologías utilizando el procedimiento DPLL:

a) $p \to (q \to p)$
b) $(p \wedge (p \to q)) \to q$
c) $((p \to q) \wedge \neg q) \to \neg p$
d) $((p \to q) \wedge \neg q) \to \neg q$

Vamos a resolver todos los apartados de este ejercicio de la misma manera: para ver que F es una tautología, vamos a demostrar que $\neg F$ es insatisfactible aplicando el procedimiento DPLL a una CNF de $\neg F$:

a) Como
$$\neg(p \to (q \to p)) \equiv p \wedge \neg(q \to p) \equiv p \wedge q \wedge \neg p,$$
el conjunto de cláusulas del cual queremos estudiar la satisfactibilidad es
$$\begin{array}{ll} 1. & p \\ 2. & q \\ 3. & \bar{p} \end{array}$$
Entonces tenemos: $\emptyset \implies_{p1} p \implies_{f3}$ "*Insat*"

Como finalmente hemos terminado con el estado "*Insat*", el conjunto de cláusulas es insatisfactible.

b) Como
$$\neg((p \wedge (p \to q)) \to q) \equiv p \wedge (\neg p \vee q) \wedge \neg q,$$
el conjunto de cláusulas del cual queremos estudiar la satisfactibilidad es
$$\begin{array}{ll} 1. & p \\ 2. & \bar{p} \vee q \\ 3. & \bar{q} \end{array}$$
Entonces tenemos: $\emptyset \implies_{p1} p \implies_{p2} p\,q \implies_{f3}$ "*Insat*"

Como finalmente hemos terminado con el estado "*Insat*", el conjunto de cláusulas es insatisfactible.

c) Como
$$\neg(((p \to q) \wedge \neg q) \to \neg p) \equiv (\neg p \vee q) \wedge \neg q \wedge p$$
el conjunto de cláusulas del cual queremos estudiar la satisfactibilidad es
$$\begin{array}{ll} 1. & \bar{p} \vee q \\ 2. & \bar{q} \\ 3. & p \end{array}$$
Como ya hemos visto en el apartado anterior, este conjunto de cláusulas es insatisfactible.

d) Como
$$\neg(((p \to q) \wedge \neg q) \to \neg q) \equiv (\neg p \vee q) \wedge \neg q \wedge q,$$
el conjunto de cláusulas del cual queremos estudiar la satisfactibilidad es
$$\begin{array}{ll} 1. & \bar{p} \vee q \\ 2. & \bar{q} \\ 3. & q \end{array}$$
Entonces tenemos: $\emptyset \implies_{p3} q \implies_{f2}$ "*Insat*"

Como finalmente hemos terminado con el estado "*Insat*", el conjunto de cláusulas es insatisfactible.

■

36. (dificultad 3) (*Invariantes de DPLL*) Supongamos que se aplica el procedimiento DPLL a un conjunto de cláusulas F y que se obtiene la traza $\emptyset \Longrightarrow \cdots \Longrightarrow M$, con $M \neq$ "*Insat*". Demuestra que entonces se cumplen las propiedades siguientes:

   a) Todos los símbolos proposicionales en M son símbolos proposicionales de F.

   b) M no contiene ningún literal más de una vez y no contiene p y $\neg p$ para ningún símbolo proposicional p.

   c) Si M es de la forma $N_0\ l_1^{\mathsf{d}}\ N_1\ l_2^{\mathsf{d}}\ \cdots\ l_n^{\mathsf{d}}\ N_n$, donde l_1, ..., l_n son los literales de decisión de M, entonces $F \cup \{l_1, \ldots, l_i\} \models N_i$ para cada $i = 0 \ldots n$, interpretando N_i como la conjunción de todos sus literales.

Demostraremos las tres propiedades por inducción sobre k, el número de aplicaciones de reglas:

- Si $k = 0$ entonces la traza es sencillamente $\emptyset$, y por lo tanto M también será $\emptyset$. Las dos primeras propiedades son obvias ya que M no contiene ningún símbolo proposicional. La tercera consiste en ver que $F \models \emptyset$, lo cual se cumple ya que $\emptyset$ es una tautología, por ser la conjunción de cero literales.
- Asumimos que la traza tiene k aplicaciones de reglas y las tres propiedades se cumplen para trazas con $k-1$ aplicaciones. Así pues, nuestra traza es de la forma $\emptyset \Longrightarrow \ldots \Longrightarrow M_{k-1} \Longrightarrow M_k$, con las tres propiedades cumpliéndose en M_{k-1}. Veamos por qué se cumplen también en M_k:

  a) Si en el paso de M_{k-1} a M_k hemos aplicado Propaga o Decide, entonces M_k es $M_{k-1}\ l$ o $M_{k-1}\ l^{\mathsf{d}}$ con l necesariamente símbolo proposicional de F debido a las condiciones laterales de las reglas. Si hemos aplicado Backtrack entonces claramente M_k sólo contiene un subconjunto de los símbolos proposicionales de M_{k-1}, que por hipótesis de inducción son todos de F.

  b) Si en M_{k-1} se cumple la propiedad, la aplicación de Propaga y Decide no la puede romper debido a la condiciones laterales de la regla. En una aplicación de Backtrack, donde M_{k-1} es $M\ l^{\mathsf{d}}\ N$ y M_k es $M\ \neg l$, la propiedad se sigue cumpliendo debido a que, por hipótesis de inducción, ni l ni $\neg l$ aparecen en M.

  c) Este es el caso más complicado. Distingamos según la regla que se ha aplicado en el último paso:

  - Propaga: en este caso, M_{k-1} es de la forma $N_0\ l_1^{\mathsf{d}}\ N_1\ l_2^{\mathsf{d}}\ \cdots\ l_n^{\mathsf{d}}\ N_n$, mientras que M_k es $N_0\ l_1^{\mathsf{d}}\ N_1\ l_2^{\mathsf{d}}\ \cdots\ l_n^{\mathsf{d}}\ N_n\ l$. Es fácil ver que sólo nos queda por demostrar que $F \cup \{l_1, \ldots, l_n\} \models l$. Para ello utilizamos que, debido a la condición lateral de la regla, existe una cláusula $l \vee C$ en F con $M_{k-1} \models \neg C$. Por hipótesis de inducción sabemos que $F \cup \{l_1, \ldots, l_n\} \models M_{k-1}$ y por lo tanto $F \cup \{l_1, \ldots, l_n\} \models \neg C$. También sabemos que, por ser $l \vee C$ una cláusula de F, se cumple $F \models l \vee C$. Por lo tanto, $F \cup \{l_1, \ldots, l_n\} \models l$.
  - Decide: en este caso, M_{k-1} es de la forma $N_0\ l_1^{\mathsf{d}}\ N_1\ l_2^{\mathsf{d}}\ \cdots\ l_n^{\mathsf{d}}\ N_n$, mientras que M_k es $N_0\ l_1^{\mathsf{d}}\ N_1\ l_2^{\mathsf{d}}\ \cdots\ l_n^{\mathsf{d}}\ N_n\ l^{\mathsf{d}}$, siendo l un literal de decisión. No hay nada a demostrar.
  - Backtrack : en este caso, M_{k-1} es de la forma $N_0\ l_1^{\mathsf{d}}\ N_1\ l_2^{\mathsf{d}}\ \cdots\ N_{n-1}\ l_n^{\mathsf{d}}\ N_n$, mientras que M_k es $N_0\ l_1^{\mathsf{d}}\ N_1\ l_2^{\mathsf{d}}\ \cdots\ N_{n-1}\ \neg l_n$, donde ahora $\neg l_n$ no es literal de decisión. Basta demostrar que $F \cup \{l_1, \ldots, l_{n-1}\} \models \neg l_n$. Para ello sabemos que existe una cláusula C de F tal que $M_{k-1} \models \neg C$ (ya que hemos aplicado Backtrack); pero como $F \cup \{l_1, \ldots, l_n\} \models M_{k-1}$ entonces $F \cup \{l_1, \ldots, l_n\} \models \neg C$. Por lo tanto $F \cup \{l_1, \ldots, l_n\} \cup \{C\}$ es insatisfactible; y como C es cláusula de F, tenemos que $F \cup \{l_1, \ldots, l_n\}$ es insatisfactible. Luego $F \cup \{l_1, \ldots, l_{n-1}\} \models \neg l_n$.

■

37. (dificultad 3) (*Corrección y completitud del procedimiento DPLL*) Supongamos que se aplica el procedimiento DPLL a un conjunto de cláusulas F, y que se obtiene la traza $\emptyset \Longrightarrow \cdots \Longrightarrow M$, a partir de donde ya no se puede aplicar más ninguna de las reglas Propaga, Decide, Falla o Backtrack. Utilizando los invariantes de DPLL, demuestra que:

    a) Si $M =$ "*Insat*" entonces F es insatisfactible.

    b) Si $M \neq$ "*Insat*" entonces $M \models F$ (y en particular F es satisfactible).

38. (dificultad 4) (*Terminación del procedimiento DPLL*) Supongamos que se aplica el procedimiento DPLL a un conjunto de cláusulas. Demuestra que no existen secuencias infinitas de la forma $\emptyset \Longrightarrow \cdots$

Para demostrar la terminación del procedimiento, vamos a hacer corresponder a cada modelo parcial un número natural, y vamos a ver que, a cada aplicación de una regla, este número decrece. La idea es reemplazar cada literal decidido por un 2, cada literal no decidido por un 1, y añadir a la derecha tantos 3 como símbolos de predicado de la fórmula no aparecen en el modelo parcial.

Sea m el número de símbolos de predicado de la fórmula F. Entonces, dado un modelo parcial M de la forma $N_0\ l_1^{\mathsf{d}}\ N_1\ l_2^{\mathsf{d}}\ N_2\ \cdots\ l_n^{\mathsf{d}}\ N_n$, donde l_1, ..., l_n son los literales de decisión de M, le asignamos el número de m dígitos

$$\mu(M) = \overbrace{1\cdots1}^{|N_0|}\ 2\ \overbrace{1\cdots1}^{|N_1|}\ 2\ \overbrace{1\cdots1}^{|N_2|}\ \cdots\ 2\ \overbrace{1\cdots1}^{|N_n|}\ \overbrace{3\cdots3}^{m-|M|},$$

donde $|N|$ representa el número de literales de la secuencia de literales N.

Basta con considerar las reglas Decide, Propaga y Backtrack, ya que si Falla se aplica obviamente la ejecución termina. Sean M, M' modelos parciales tales que $M \Longrightarrow M'$. Suponemos que M es de la forma $N_0\ l_1^{\mathsf{d}}\ N_1\ \cdots\ l_{n-1}^{\mathsf{d}}\ N_{n-1}\ l_n^{\mathsf{d}}\ N_n$, y distinguimos varios casos:

- Decide: Entonces M' es $N_0\ l_1^{\mathsf{d}}\ N_1\ \cdots\ l_{n-1}^{\mathsf{d}}\ N_{n-1}\ l_n^{\mathsf{d}}\ N_n\ l^{\mathsf{d}}$, donde l es el nuevo literal de decisión. Como

$$\begin{aligned}
\mu(M) &= \overbrace{1\cdots1}^{|N_0|}\ 2\ \overbrace{1\cdots1}^{|N_1|}\ \cdots\ 2\ \overbrace{1\cdots1}^{|N_{n-1}|}\ 2\overbrace{1\cdots1}^{|N_n|}\ \overbrace{\underline{3}\cdots3}^{m-|M|}\\
\mu(M') &= \overbrace{1\cdots1}^{|N_0|}\ 2\ \overbrace{1\cdots1}^{|N_1|}\ \cdots\ 2\ \overbrace{1\cdots1}^{|N_{n-1}|}\ 2\overbrace{1\cdots1}^{|N_n|}\ \underline{2}\ \overbrace{3\cdots3}^{m-|M|-1}
\end{aligned}$$

  (se ha subrayado el primer dígito diferente por la izquierda) resulta que $\mu(M) > \mu(M')$.

- Propaga: Entonces M' es $N_0\ l_1^{\mathsf{d}}\ N_1\ \cdots\ l_{n-1}^{\mathsf{d}}\ N_{n-1}\ l_n^{\mathsf{d}}\ N_n\ l$, donde l es el literal propagado. Como

$$\begin{aligned}
\mu(M) &= \overbrace{1\cdots1}^{|N_0|}\ 2\ \overbrace{1\cdots1}^{|N_1|}\ \cdots\ 2\ \overbrace{1\cdots1}^{|N_{n-1}|}\ 2\overbrace{1\cdots1}^{|N_n|}\ \overbrace{\underline{3}\cdots3}^{m-|M|}\\
\mu(M') &= \overbrace{1\cdots1}^{|N_0|}\ 2\ \overbrace{1\cdots1}^{|N_1|}\ \cdots\ 2\ \overbrace{1\cdots1}^{|N_{n-1}|}\ 2\overbrace{1\cdots1}^{|N_n|}\ \underline{1}\ \overbrace{3\cdots3}^{m-|M|-1}
\end{aligned}$$

  resulta que $\mu(M) > \mu(M')$.

- Backtrack: Entonces M' es $N_0\ l_1^{\mathsf{d}}\ N_1\ \cdots\ l_{n-1}^{\mathsf{d}}\ N_{n-1}\ \neg l_n$, donde ahora $\neg l_n$ no es literal de decisión. Como

$$\begin{aligned}
\mu(M) &= \overbrace{1\cdots1}^{|N_0|}\ 2\ \overbrace{1\cdots1}^{|N_1|}\ \cdots\ 2\ \overbrace{1\cdots1}^{|N_{n-1}|}\ \underline{2}\ \overbrace{1\cdots1}^{|N_n|}\ \overbrace{3\cdots3}^{m-|M|}\\
\mu(M') &= \overbrace{1\cdots1}^{|N_0|}\ 2\ \overbrace{1\cdots1}^{|N_1|}\ \cdots\ 2\ \overbrace{1\cdots1}^{|N_{n-1}|}\ \underline{1}\ \overbrace{3\cdots3}^{|N_n|+m-|M|}
\end{aligned}$$

  resulta que $\mu(M) > \mu(M')$.

■

39. (dificultad 3) El problema de *AllSAT* consiste en obtener todos los modelos de una fórmula proposicional F. Desarrolla un algoritmo para *AllSAT* utilizando llamadas independientes al procedimiento DPLL.

La idea es muy sencilla. Tomemos la fórmula F y llamamos al procedimiento DPLL sobre F. Si nos dice que F es insatisfactible, entonces F no tiene ningún modelo y terminamos. En caso contrario el procedimiento nos devuelve un modelo I. Ahora queremos forzar que DPLL nos devuelva un modelo distinto a I. Lo que hacemos es considerar I como una conjunción de literales, es decir, la conjunción de todos los literales que son ciertos en I. Si ahora tomamos la fórmula $F \wedge \neg I$ es fácil ver que es una fórmula en CNF que tiene exactamente los mismos modelos que F menos I, que queda prohibido debido a la cláusula $\neg I$. Finalmente, el proceso se deba iterar, añadiendo nuevas cláusulas para cada modelo encontrado hasta que la fórmula así obtenida se vuelva insatisfactible, lo que significará que ya hemos enumerado todos los modelos:

mientras F satisfactible

Sea I modelo de F obtenido llamando DPLL sobre F

$F := F \wedge \neg I$

fmientras

■