Lógica Clausular Proposicional
Lógica Computacional 2020-II, Nota de clase 3

Favio Ezequiel Miranda Perea Araceli Liliana Reyes Cabello
Lourdes Del Carmen González Huesca Pilar Selene Linares Arévalo

Facultad de Ciencias UNAM

Las cláusulas son una clase especial de fórmulas relevantes en distintas aplicaciones dentro y fuera de la lógica. En esta nota discutimos la transformación de cualquier fórmula a la llamada forma clausular. En notas posteriores se abordará el famoso problema de satisfacibilidad SAT de gran importancia en la teoría de la complejidad computacional y la regla de resolución binaria de Robinson, piedra angular de la programación lógica que utilizan la representación clausular de fórmulas.

Para llegar a la forma clausular de una fórmula se utilizan transformaciones de fórmulas mediante equivalencias lógicas llamadas formas normales, que estudiamos a continuación.

1. Forma Normal Negativa

El objetivo de esta forma normal es obtener una fórmula equivalente a una fórmula dada en donde no figuren implicaciones ni bicondicionales además en donde los símbolos de negación sólo afectan a fórmulas atómicas.

Definición 1. *Una fórmula φ está en forma normal negativa si y sólo si se cumplen las siguientes:*

1. *φ no contiene ni equivalencias ni implicaciones;*
2. *las negaciones que figuran en φ afectan sólo a fórmulas atómicas.*

La transformación a forma normal negativa se sirve de las siguientes equivalencias.

Proposición 1 (Leyes de la Negación)**.** *Se cumplen las siguientes equivalencias lógicas.*

- *Doble Negación:* $\neg\neg\varphi \equiv \varphi$.
- *De Morgan:* $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$.
- *De Morgan:* $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$.
- $\neg(\varphi \to \psi) \equiv \varphi \wedge \neg\psi$
- $\neg(\varphi \leftrightarrow \psi) \equiv \neg\varphi \leftrightarrow \psi \equiv \varphi \leftrightarrow \neg\psi$.

Demostración. Se deja como ejercicio. □

Proposición 2. *Sea φ una fórmula. Podemos encontrar de manera algorítmica una fórmula ψ lógicamente equivalente a φ y tal que ψ está en forma normal negativa. En tal caso ψ se denota con $fnn(\psi)$.*

Demostración. Dada φ, eliminar los símbolos $\to, \leftrightarrow$ mediante las equivalencias usuales, es decir:

- Eliminar bicondicionales usando que $\varphi \leftrightarrow \psi \equiv (\varphi \to \psi) \wedge (\psi \to \varphi)$.
- Eliminar implicaciones usando que $\varphi \to \psi \equiv \neg\varphi \vee \psi$

Posteriormente

- Introducir las negaciones de manera que sólo afecten a literales mediante:
  $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi \qquad \neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$
- Eliminar las dobles negaciones mediante $\neg\neg\varphi \equiv \varphi$.

Es decir se aplicarán exhaustivamente las leyes de negación de manera adecuada. Así la fórmula ψ obtenida es $fnn(\varphi)$. □

1.1. Implementación

Se deja como ejercicio implementar la función `fnn: Prop -> Prop` que transforma una fórmula φ en su forma normal negativa. Las siguientes observaciones son de importancia:

1. Podemos suponer que la fórmula de entrada no tiene implicaciones ni equivalencias (recordar las funciones que eliminan estos conectivos en la nota 2). Esto deja las siguientes clases de fórmulas: atómicas, conjunciones, disyunciones y las negaciones de cada una de estas.
2. Si la fórmula no es una negación entonces: si es atómica ya está en `fnn` y en otro caso la `fnn` se calcula *recursivamente* en los operandos respetando el conectivo principal.
3. Si la fórmula es una negación debe hacerse un análisis de casos:
   a) si es negación de atómica ya está en `fnn`;
   b) si es negación de negación, entonces ambas negaciones se eliminan y se llama recursivamente a la función `fnn`;
   c) si es negación de conjunción o de disyunción se aplican las leyes de De Morgan haciendo las llamadas recursivas donde corresponda.

2. Forma Normal Conjuntiva

La llamada *forma normal conjuntiva* permite expresar cualquier fórmula proposicional como una conjunción de disyunciones llamadas **cláusulas**. resolución

Definición 2. *Una literal ℓ es una fórmula atómica (variable proposicional p, $\bot$ o $\top$) o la negación de una fórmula atómica.*

Una literal es **negativa** si es una negación, en otro caso decimos que es **positiva**.

Definición 3. *Dada una literal ℓ definimos su literal contraria, denotada ℓ^c, como sigue:*

$$\ell^c = \begin{cases} a & si \quad \ell = \neg a \\ \neg a & si \quad \ell = a \end{cases}$$

Donde a es una fórmula atómica, es decir, una variable proposicional, $\top$ o $\bot$.
El par $\{\ell, \ell^c\}$ se llama un par de literales complementarias.

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

Definición 4. *Una cláusula $\mathcal{C}$ es una literal o una disyunción de literales.*

Definición 5. *Una fórmula φ está en* ***forma normal conjuntiva*** `fnc` *si y sólo si es de la forma $\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \ldots \wedge \mathcal{C}_n$ donde para cada $\mathcal{C}_i$ es una cláusula.*

En particular se sigue que cualquier literal y cualquier cláusula están en forma normal conjuntiva. Los conceptos anteriores de literal, cláusula y forma normal conjuntiva se definen de manera breve mediante la siguientes gramática:

$$\begin{array}{rcll} a & ::= & \bot \mid \top \mid p & \text{– atómicas} \\ \ell & ::= & a \mid \neg\, a & \text{– literales} \\ \mathcal{C} & ::= & \ell \mid \ell \vee \mathcal{C} & \text{– cláusulas} \\ F & ::= & \mathcal{C} \mid \mathcal{C} \wedge F & \text{– formas normales conjuntivas} \end{array}$$

Proposición 3. *Cualquier fórmula proposicional φ puede transformarse algorítmicamente en una fórmula ψ tal que $\varphi \equiv \psi$ y ψ está en forma normal conjuntiva. En este caso ψ se denota con $fnc(\varphi)$.*

Demostración. Basta aplicar las siguientes transformaciones en orden:

- Obtener la forma normal negativa correspondiente de φ.
- Reescribir las conjunciones y disyunciones mediante las leyes distributivas:

$$\begin{array}{l} (\varphi \wedge \psi) \vee (\varphi \wedge \chi) \equiv \varphi \wedge (\psi \vee \chi) \\ (\varphi \vee \psi) \wedge (\varphi \vee \chi) \equiv \varphi \vee (\psi \wedge \chi) \end{array}$$

  Cuyo caso general es

$$\begin{array}{rll} (\alpha_1 \vee \ldots \vee \alpha_n) \wedge (\beta_1 \vee \ldots \vee \beta_m) \equiv & (\alpha_1 \wedge \beta_1) \vee (\alpha_1 \wedge \beta_2) \vee \ldots \vee (\alpha_1 \wedge \beta_m) & \vee \\ & (\alpha_2 \wedge \beta_1) \vee (\alpha_2 \wedge \beta_2) \vee \ldots \vee (\alpha_3 \wedge \beta_m) & \vee \\ & \vdots & \\ & (\alpha_n \wedge \beta_1) \vee (\alpha_n \wedge \beta_2) \vee \ldots \vee (\alpha_n \wedge \beta_m) & \end{array}$$

- **Opcionalmente** se puede simplificar usando

$$\begin{array}{lll} \neg\varphi \vee \varphi \equiv \top & \top \vee \varphi \equiv \top & \top \wedge \varphi \equiv \varphi \\ \bot \vee \varphi \equiv \varphi & \bot \wedge \varphi \equiv \bot & \varphi \vee \varphi \equiv \varphi \end{array}$$

□

Ejemplo 2.1. Obtener la forma normal conjuntiva de $\varphi = (\neg p \to q) \to (\neg r \to s)$.
Aplicando el método recién descrito obtenemos:

$$\begin{array}{rcl} (\neg p \to q) \to (\neg r \to s) & \equiv & \neg(\neg\neg p \vee q) \vee (\neg\neg r \vee s) \\ & \equiv & (\neg p \wedge \neg q) \vee (r \vee s) \\ & \equiv & (\neg p \vee r \vee s) \wedge (\neg q \vee r \vee s) \\ & = & fnc(\varphi) \end{array}$$

2.1. Implementación de la transformación a la forma normal conjuntiva

Suponemos que las fórmulas de entrada ya están en forma normal negativa por lo que las únicas fórmulas a considerar son literales, conjunciones y disyunciones.

La función `fnc:: Prop -> Prop` que lleva a cabo la transformación deseada debe definirse *recursivamente* de acuerdo a las siguientes observaciones:

- Si la fórmula de entrada es una literal ya está en forma normal conjuntiva y simplemente se devuelve como salida.
- Si la fórmula es una conjunción $\varphi_1 \wedge \varphi_2$ entonces se llama recursivamente a la función `fnc` en los operandos de la conjunción, devolviendo $\texttt{fnc}(\varphi_1) \wedge \texttt{fnc}(\varphi_2)$
- Si la fórmula es una disyunción $\varphi_1 \vee \varphi_2$ entonces se llama recursivamente a la función `fnc` en los operandos de la disyunción pero no podemos devolver $\texttt{fnc}(\varphi_1) \vee \texttt{fnc}(\varphi_2)$ pues esta fórmula no está en forma normal conjuntiva (por ser una disyunción). Por lo que tenemos que transformar esta fórmula en una conjunción mediante las leyes de distributividad.
  Para esto tenemos que definir una función auxiliar `distr:: Prop -> Prop -> Prop` que aplique adecuadamente las leyes distributivas a las formas normales conjuntivas de φ_1 y φ_2. De esta manera la fórmula que `fnc` devuelve en el caso de la disyunción será $\texttt{distr(fnc}(\varphi_1)\texttt{,fnc}(\varphi_2)\texttt{)}$

Para implementar la función auxiliar `distr:: Prop -> Prop -> Prop` son útiles las siguientes observaciones:

- Podemos suponer que las fórmulas de entrada φ_1 y φ_2 son formas conjuntivas.
- Si ambas φ_1, φ_2 son literales entonces devolvemos $\varphi_1 \vee \varphi_2$ pues en este caso tal disyunción es una cláusula y por lo tanto una forma conjuntiva.
- Si $\varphi_1 = \varphi_{11} \wedge \varphi_{12}$ entonces aplicamos distributividad:

  $$\varphi_1 \vee \varphi_2 = (\varphi_{11} \wedge \varphi_{12}) \vee \varphi_2 \equiv (\varphi_{11} \vee \varphi_2) \wedge (\varphi_{12} \vee \varphi_2)$$

  pero para el resultado debemos también aplicar recursivamente la función `distr` a los pares φ_{11}, φ_2 y φ_{12}, φ_2.
- Si $\varphi_2 = \varphi_{21} \wedge \varphi_{22}$ la definición es análoga al caso anterior.

Obsérvese que la definición es *no determinista* para el caso en que ambas fórmulas de entrada sean conjunciones.

2.2. Validez y satisfacibilidad mediante formas normales conjuntivas

El empleo de las formas normales conjuntivas simplifica el procedimiento para decidir si una fórmula dada es válida, es decir es tautología.

Proposición 4. *Una cláusula $\mathcal{C} = \ell_1 \vee \ell_2 \vee \ldots \vee \ell_n$ es tautología ($\models \varphi$) si y sólo si existen $1 \leqslant i, j \leqslant n$ tales que $\ell_i^c = \ell_j$. Es decir, $\models \mathcal{C}$ si y sólo si $\mathcal{C}$ contiene un par de literales complementarias.*

Demostración. Se deja como ejercicio. □

La proposición anterior permite tener un algoritmo para verificar si $\models \varphi$, cuando φ está en forma normal conjuntiva, digamos $\varphi = \mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_n$:

1. Para cada $1 \leqslant i \leqslant n$, buscar en $\mathcal{C}_i$ un par de literales complementarias.
2. Si tal par existe para cada cláusula $\mathcal{C}_i$ entonces $\models \varphi$.
3. En otro caso $\not\models \varphi$, es decir φ no es tautología.

Ahora bien, recordando el principio de refutación

$$\models \varphi \text{ si y sólo si } \neg\varphi \text{ es no satisfacible}$$

podemos también usar el algoritmo anterior para decidir la satisfacibilidad de una fórmula puesto que, de acuerdo al principio anterior, φ es satisfacible si y sólo si $\not\models \neg\varphi$, es decir, si y sólo si $\neg\varphi$ no es tautología. Obsérvese que este algoritmo es sólo para decidir satisfacibilidad y por si solo no proporciona un estado $\mathcal{I}$ que satisfaga a φ.
Se deja como ejercicio mostrar la correctud del siguiente argumento mediante el algoritmo anterior:

Si el dragón se enfada, el caballero Lancelot quedará paralizado del susto,
pero si se queda paralizado del susto, entonces
no puede sino apelar a la bondad del dragón y así no ser engullido.
Luego entonces, si el dragón se enfada,
el caballero Lancelot tendrá que apelar a su bondad o será engullido

3. Resolución Proposicional

Para terminar nuestro panorama de la lógica clausular presentamos la regla de resolución binaria de Robinson y en particular un nuevo método para decidir la correctud de un argumento lógico. Más adelante estudiaremos a profundidad la versión de esta regla para la lógica de predicados y su relevancia en la programación lógica. Las teorías de prueba basadas en resolución han demostrado ser completas tanto para lógica proposicional como para lógica de predicados. La aportación de Robinson [1], es dar una versión generalizada de la regla de inferencia de resolución sobre cláusulas.

Definición 6. *Sean $\mathcal{C}_1$, $\mathcal{C}_2$ cláusulas y ℓ una literal. La regla de inferencia conocida como resolución binaria proposicional se define como sigue:*

$$\frac{\mathcal{C}_1 \vee \ell \qquad \ell^c \vee \mathcal{C}_2}{\mathcal{C}_1 \vee \mathcal{C}_2}\ \textsc{Res}$$

*donde ℓ^c es la literal contraria de ℓ. En tal situación decimos que se **resuelven** las dos premisas con respecto a la literal ℓ y a la cláusula resultante $\mathcal{C}_1 \vee \mathcal{C}_2$ se le llama resolvente o resolvente binario.*

El adjetivo *binaria* se refiere al hecho de que tenemos dos premisas, la regla toma dos cláusulas tales que existe una literal ℓ que figura en una de ellas mientras que la literal contraria ℓ^c figura en la otra. Si bien en la definición de la regla las literales ℓ y ℓ^c aparecen al final y principio de las cláusulas respectivamente, el orden no importa dado que la disyunción es conmutativa.
Veamos un par de ejemplos:

$$\frac{\neg p \vee q \vee r \qquad s \vee \neg r \vee \neg t}{\neg p \vee q \vee s \vee \neg t} \qquad\qquad \frac{t \vee \neg s \vee q \qquad \neg q \vee w \vee s \vee u}{t \vee \neg s \vee w \vee s \vee u}$$

Obsérvese que la regla es **no determinista**, en el segundo ejemplo otro resolvente posible es $t \vee q \vee \neg q \vee w \vee u$.

Dado que las literales también son cláusulas la aplicación de resolución a p y $\neg p$ devuelve como resultado la llamada **cláusula vacía**, denotada $\square$, es decir la siguiente es una instancia válida de resolución

$$\frac{p \qquad \neg p}{\square}$$

[1] Robinson, J. Alan (1965). "A Machine-Oriented Logic Based on the Resolution Principle". Journal of the ACM vol. 12 doi:10.1145/321250.321253.

Es importante observar que la regla opera sólamente sobre un par de literales complementarias a la vez y nunca con más, por ejemplo la siguiente es una aplicación **incorrecta** de resolución:

$$\frac{p \vee \neg q \qquad q \vee \neg p}{\square}$$

Aplicaciones correctas dan como resultado los resolventes totalmente diferentes a saber $p \vee \neg p$ y $\neg q \vee q$. En particular, a partir de las premisas dadas jamás se podrá generar la cláusula vacía.

La resolución binaria proporciona un método de decisión para la lógica que, al igual que en el caso de los tableaux semánticos, utiliza el principio de **refutación** para decidir la consecuencia lógica:

para decidir si $\Gamma \models \varphi$ *basta demostrar que* $\Gamma \cup \{\neg\varphi\}$ *es insatisfacible*

En resolución binaria basta con obtener la cláusula vacía usando la regla de resolución binaria repetidas veces, partir del conjunto de cláusulas de la formas normales conjuntivas del conjunto $\Gamma \cup \{\neg\varphi\}$. Este proceso se conoce como una *refutación* del conjunto de cláusulas.

Ejemplo 3.1. Veamos un ejemplo sencillo, queremos decidir si

$$\{p \to (q \to r), \neg(q \to r)\} \models \neg p$$

El conjunto de formas normales conjuntivas para $\Gamma \cup \{\neg\neg p\}$ es:

$$\{\neg p \vee \neg q \vee r, q \wedge \neg r, p\}$$

a partir de este se obtiene el conjunto de cláusulas correspondiente al separar las cláusulas de cada forma normal conjuntiva

$$\{\neg p \vee \neg q \vee r, q, \neg r, p\}$$

Finalmente de este último conjunto obtenemos la siguiente **derivación** de la cláusula vacía $\square$:

1.	$\neg p \vee \neg q \vee r$	$Hip.$
2.	q	$Hip.$
3.	$\neg r$	$Hip.$
4.	p	$Hip.$
5.	$\neg p \vee r$	$Res(1,2)$
6.	r	$Res(5,4)$
7.	$\square$	$Res(6,3)$

Ejemplo 3.2. Veamos otro ejemplo, queremos decidir si el argumento

$$p \to q,\ r \to s,\ p \vee r \ /\ \therefore q \vee s$$

es correcto. El conjunto a refutar es:

$$\{\neg p \vee q,\ \neg r \vee s,\ p \vee r,\ \neg q,\ \neg s\}$$

Obtenemos la siguiente derivación de $\square$:

1.	$\neg p \vee q$	$Hip.$
2.	$\neg r \vee s$	$Hip.$
3.	$p \vee r$	$Hip.$
4.	$\neg q$	$Hip.$
5.	$\neg s$	$Hip.$
6.	$\neg p$	$Res(1,4)$
7.	r	$Res(6,3)$
8.	s	$Res(2,7)$
9.	$\square$	$Res(5,8)$

De manera que el argumento es correcto.

4. Algoritmos de saturación

Definición 7. *Si $\mathbb{S}$ es cualquier conjunto de cláusulas entonces la resolución de $\mathbb{S}$, denotada $\mathcal{R}(\mathbb{S})$ es el conjunto que consiste de $\mathbb{S}$ junto con* ***todos*** *los resolventes de cláusulas de $\mathbb{S}$, es decir*

$$\mathcal{R}(\mathbb{S}) = \mathbb{S} \cup \{E \mid \text{ existen } C, D \in \mathbb{S} \text{ tales que } E \text{ es un resolvente de } C \text{ y } D\}$$

Es importante recalcar que dentro de los resolventes también se incluyen aquellos obtenidos mediante la eliminación de literales repetidas. Por ejemplo $p \vee q \vee p$ es un resolvente de $p \vee q \vee \neg s$ y $s \vee p$, por lo que $q \vee p$ y $p \vee q$ también se consideran resolventes. En particular si $r \vee r$ es un resolvente entonces r también lo es.

Definición 8. *La n-ésima resolución de $\mathbb{S}$ se define recursivamente como sigue:*

$$\begin{aligned} Res_0(\mathbb{S}) &= \mathbb{S} \\ Res_{n+1}(\mathbb{S}) &= \mathcal{R}(Res_n(\mathbb{S})) \end{aligned}$$

La importancia de la n-ésima resolución se da en la siguiente proposición que nos da una herramienta muy útil para la inferencia lógica:

Proposición 5. *Sea $\mathbb{S}$ es un conjunto finito de cláusulas. Entonces $\mathbb{S}$ es no satisfacible si y sólo si $\square \in Res_n(\mathbb{S})$ para alguna $n \in \mathbb{N}$.*

El resultado anterior proporciona, en teoría, un método para verificar si un conjunto de cláusulas $\mathbb{S}$ es insatisfacible, basta construir los conjuntos $Res_n(\mathbb{S})$ hasta hallar $\square$. Esto se hace mediante los llamados algoritmos de saturación que se encargan de generar todas las derivaciones con resolución a partir de un conjunto dado $\mathbb{S}$.
Por supuesto, el método por si sólo es demasiado ineficiente pero combinado con algoritmos de poda del espacio de búsqueda y de eliminación de cláusulas redundantes se obtiene un mecanismo poderoso de inferencia.
Analicemos los escenarios posibles, en teoría hay tres:

1. En algún momento $\square$ es generada, es decir $\square \in Res_n(\mathbb{S})$ para algún $n \in \mathbb{N}$.
   En este caso el conjunto Γ de entrada es insatisfacible.

2. El algoritmo termina sin generar $\square$ jamás, es decir, en algún momento se tiene $Res_n(\mathbb{S}) = Res_{n+1}(\mathbb{S})$ por lo que no hay más resolventes posibles, pero $\square \notin Res_n(\mathbb{S})$.
   En este caso Γ es satisfacible.

3. La saturación no termina nunca pero tampoco se genera la cláusula vacía, es decir, se tiene la certeza de que para cualquier $n \in \mathbb{N}$, $\square \notin Res_n(\mathbb{S})$.
   En este caso Γ es satisfacible.

En la práctica el tercer escenario es poco factible y en su lugar tenemos

3'. La saturación no termina, el algoritmo se ejecuta hasta agotar los recursos computacionales pero no se genera $\square$.
   En este caso se **desconoce** si Γ es satisfacible.

Ejemplo 4.1. Sea $\mathbb{S}$ el conjunto que consta de las siguientes cláusulas:

$$\begin{aligned} &1. \quad p \vee r \\ &2. \quad \neg p \vee \neg r \\ &3. \quad p \\ &4. \quad r \end{aligned}$$

Empezamos con $Res_0(\mathbb{S}) = \mathbb{S}$.
Entonces, $Res_1(\mathbb{S}) = \mathcal{R}(Res_0(\mathbb{S})) = Res_0(\mathbb{S}) \cup$

$$\begin{array}{rrl} 5. & \neg r \vee r & res(1,2) \\ 6. & \neg r & res(2,3). \\ 7. & \neg p & res(2,4). \end{array}$$

Y finalmente: $Res_2(\mathbb{S}) = \mathcal{R}(Res_1(\mathbb{S})) = Res_1(\mathbb{S}) \cup$

$$\begin{array}{rrl} 8. & p \vee r & res(1,5). \\ 9. & p & res(1,6). \\ 10. & r & res(1,7). \\ 11. & \neg p \vee \neg r & res(2,5). \\ 12. & \square & res(3,7). \end{array}$$

Por lo tanto $\square \in Res_2(\mathbb{S})$ y $\mathbb{S}$ no es satisfacible.

Ejemplo 4.2. Sea $\mathbb{S}$ el conjunto que consta de las siguientes cláusulas:

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

Comenzamos con $Res_0(\mathbb{S}) = \mathbb{S}$.
La primera iteración genera: $Res_1(\mathbb{S}) = \mathcal{R}(Res_0(\mathbb{S})) = Res_0(\mathbb{S}) \cup$

$$\begin{array}{rrl} 5. & \neg q & res(1,2). \\ 6. & \neg r & res(1,3). \\ 7. & p \vee r & res(2,4). \\ 8. & p \vee q & res(3,4). \end{array}$$

La siguiente iteración $Res_2(\mathbb{S}) = \mathcal{R}(Res_1(\mathbb{S})) = \ Res_1(\mathbb{S}) \cup$

$$\begin{array}{rrl} 9. & r & res(1,7). \\ 10. & q & res(1,8). \\ 11. & p \vee p & res(2,8). \\ 12. & p \vee p & res(3,7). \\ 13. & r & res(4,5). \\ 14. & q & res(4,6). \\ 15. & p & res(5,8). \\ 16. & p & res(6,7). \end{array}$$

Y finalmente $Res_3(\mathbb{S}) = \mathcal{R}(Res_2(\mathbb{S})) = Res_2(\mathbb{S}) \cup$

$$\begin{array}{rrl} 17. & p & res(1,11). \\ 18. & p & res(1,12). \\ 19. & \square & res(1,15). \end{array}$$

Por lo tanto $\square$ pertenece a la tercera resolución de $\mathbb{S}$, así que $\mathbb{S}$ es no satisfacible.

Ejemplo 4.3. Sea $\mathbb{S}$ el conjunto que consta de las siguientes cláusulas:

$$\begin{array}{ll} 1. & r \vee p \vee q \\ 2. & r \\ 3. & \neg q \end{array}$$

Comenzamos con $Res_0(\mathbb{S}) = \mathbb{S}$.
Después $Res_1(\mathbb{S}) = \mathcal{R}(Res_0(\mathbb{S})) = Res_0(\mathbb{S}) \cup$

$$4. \quad r \vee p \quad res(1,3)$$

Pero al calcular la siguiente iteración nos encontramos con $Res_2(\mathbb{S}) = \mathcal{R}(Res_1(\mathbb{S})) = Res_1(\mathbb{S}) \cup \varnothing$
Por lo tanto $Res_n(\mathbb{S}) = Res_1(\mathbb{S})$ para toda $n \geqslant 1$ y como $\square \notin Res_1(\mathbb{S})$ entonces $\mathbb{S}$ es satisfacible.

Los siguientes ejercicios para decidir la correctud de argumentos dejan ver varios problemas en la búsqueda de la cláusula vacía, los cuales complican el proceso de mecanización discutido arriba. Los sistemas que implementan el cálculo de resolventes binarios deben servirse de diversas *heurísticas* para podar el espacio de búsqueda. El área de la lógica que se dedica a estos menesteres es, nuevamente, el razonamiento automatizado.

- $p \vee (q \wedge r),\ p \to q,\ q \leftrightarrow s,\ / \therefore\ q \wedge s$
- $\neg(\neg p \to q),\ \neg(r \leftrightarrow p),\ p \vee r,\ \neg(r \to q)\ / \therefore\ \neg(p \to q)$
- $p \to q,\ r \to s,\ p \vee r,\ \neg(q \wedge s)\ / \therefore\ (q \to p) \wedge (s \to r)$