

Lógica Clausular Proposicional

Lógica Computacional 2016-2, Nota de clase 3

Favio Ezequiel Miranda Perea Araceli Liliana Reyes Cabello
Lourdes Del Carmen González Huesca

10 de febrero de 2016

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 así como 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.

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 sin implicaciones ni bicondicionales, donde además los símbolos de negación solo afecten a fórmulas atómicas.

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

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 la siguiente

Proposición 1 (Leyes de la Negación) Se tienen 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: $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$.
- $\neg(\varphi \rightarrow \psi) \equiv \varphi \wedge \neg\psi$
- $\neg(\varphi \leftrightarrow \psi) \equiv \neg\varphi \leftrightarrow \psi \equiv \varphi \leftrightarrow \neg\psi$.

Demostración. 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 $fnn(\varphi)$.

Demostración. Dada φ , eliminar los símbolos $\rightarrow, \leftrightarrow$ mediante las equivalencias usuales y al resultado aplicarle exhaustivamente las leyes de negación de manera adecuada. La fórmula ψ así 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: si es negación de atómica ya está en `fnn`; si es negación de negación ambas negaciones se eliminan y se llama recursivamente a la función `fnn`; 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.

Definición 2 Una literal ℓ es una fórmula atómica (variable proposicional p , \perp 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:

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

Donde a es una fórmula atómica, es decir, una variable proposicional, \top o \perp .

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

$$\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \dots \wedge \mathcal{C}_n$$

donde para cada $1 \leq i \leq n$, \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 grámatica:

$$\begin{aligned} a &::= \perp \mid \top \mid p && - - \text{atómicas} \\ \ell &::= a \mid \neg a && - - \text{literales} \\ \mathcal{C} &::= \ell \mid \ell \vee \mathcal{C} && - - \text{cláusulas} \\ F &::= \mathcal{C} \mid \mathcal{C} \wedge F && - - \text{formas normales conjuntivas} \end{aligned}$$

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.
- Reescribir las conjunciones y disyunciones mediante las leyes distributivas:

$$\begin{aligned} (\varphi \wedge \psi) \vee (\varphi \wedge \chi) &\equiv \varphi \wedge (\psi \vee \chi) \\ (\varphi \vee \psi) \wedge (\varphi \vee \chi) &\equiv \varphi \vee (\psi \wedge \chi) \end{aligned}$$

- Opcionalmente se puede simplificar usando que

$$\neg\varphi \vee \varphi \equiv \top \quad \top \vee \varphi \equiv \top \quad \top \wedge \varphi \equiv \varphi \quad \perp \vee \varphi \equiv \varphi \quad \perp \wedge \varphi \equiv \perp \quad \varphi \vee \varphi \equiv \varphi$$

□

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

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

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 formulas 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 `fnc(φ_1) \wedge fnc(φ_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 `fnc(φ_1) \vee fnc(φ_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 `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 `distr(fnc(φ_1), fnc(φ_2))`

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

- Podemos suponer que las fórmulas de entrada φ_1, 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 $C = \ell_1 \vee \ell_2 \vee \dots \vee \ell_n$ es tautología ($\models \varphi$) si y sólo si existen $1 \leq i, j \leq n$ tales que $\ell_i^c = \ell_j$. Es decir, $\models C$ si y sólo si C contiene un par de literales complementarias.

Esta proposición proporciona un algoritmo para verificar si $\models \varphi$, cuando φ está en forma normal conjuntiva, digamos $\varphi = \mathcal{C}_1 \wedge \dots \wedge \mathcal{C}_n$

1. Para cada $1 \leq i \leq n$, 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. El problema SAT

El problema de satisfacibilidad para la lógica proposicional, denotado usualmente como SAT es de suma importancia en la teoría de la complejidad computacional y en general en Ciencia de la Computación. Este problema tiene aplicaciones tanto teóricas como prácticas. En particular resulto ser el primer problema NP-completo, lo cual fue probado por Cook en 1971. De hecho antes del Teorema de Cook, el concepto de NP completud ni siquiera existía.

En su forma más común el problema SAT es el siguiente: Dado un conjunto $P = \{p_1, \dots, p_n\}$ de variables proposicionales y un conjunto C de cláusulas con variables en P

¿ Existe una interpretación \mathcal{I} que satisfaga a C ?

El área de la lógica conocida como razonamiento automatizado se encarga, entre muchas otras cosas, del desarrollo de algoritmos para resolver el problema SAT sobre todo con respecto a ciertas familias de cláusulas para los que sí existen algoritmos eficientes, como son las llamadas cláusulas de Horn. Hoy en día existen programas sumamente eficientes para resolver el problema SAT, los cuales se conocen como solucionadores SAT¹ y tienen muchas aplicaciones relevantes fuera de la lógica.

Es de interés observar que si se restringe el problema de forma que cada cláusula dada tenga a lo más k -variables (el llamado problema k -SAT) sigue siendo NP-completo para $k \geq 3$. Por otra parte el caso $k = 2$ se resuelve en tiempo polinomial.

¹En inglés SAT solvers.

4. 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.

Definición 6 Sean C_1, C_2 cláusulas y ℓ una literal. La regla de inferencia conocida como resolución binaria proposicional se define como sigue:

$$\frac{C_1 \vee \ell \quad \ell^c \vee C_2}{C_1 \vee C_2} (Res)$$

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 $C_1 \vee C_2$ 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:

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

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

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

Es de suma importancia observar que la regla opera solamente sobre un par de literales complementarias a la vez y nunca con más, por ejemplo la siguiente es una aplicación INCORRECTA de resolución:

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

Aplicaciones correctas dan como resultado los resolventes $p \vee \neg p$ 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 para lo cual basta con obtener la cláusula vacía usando la regla de resolución binaria, 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.

Veamos un ejemplo sencillo, queremos decidir si

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

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 \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)*

Veamos otro ejemplo, queremos decidir si el argumento

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

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.

5. Algoritmos de saturacion

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

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

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 proporciona una herramienta muy útil para la inferencia lógica

Proposición 5 *si \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 encarga 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 proporciona un mecanismo poderoso de inferencia.

En la teoría hay tres escenarios posibles:

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 mas resolventes posibles, pero $\square \notin Res_n(\mathbb{S})$. En este caso Γ es satisfacible.
3. La saturación no termina nunca pero no 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 5.1 Sea \mathbb{S} el conjunto que consta de las siguientes cláusulas:

$$Res_0(\mathbb{S}) = \mathbb{S}$$

1. $p \vee r$
2. $\neg p \vee \neg r$
3. p
4. r

$$Res_1(\mathbb{S}) = \mathcal{R}(Res_0(\mathbb{S})) = Res_0(\mathbb{S}) \cup$$

5. $\neg r \vee r$ $res(1, 2)$
6. $\neg r$ $res(2, 3).$
7. $\neg p$ $res(2, 4).$

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

8. $p \vee r \quad res(1, 5).$
9. $p \quad res(1, 6).$
10. $r \quad res(1, 7).$
11. $\neg p \vee \neg r \quad res(2, 5).$
12. $\square \quad res(3, 7).$

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

Ejemplo 5.2 Sea \mathbb{S} el conjunto que consta de las siguientes cláusulas:

$$Res_0(\mathbb{S}) = \mathbb{S}$$

1. $\neg p$
2. $\neg q \vee p$
3. $\neg r \vee p$
4. $q \vee r$

$$Res_1(\mathbb{S}) = \mathcal{R}(Res_0(\mathbb{S})) = Res_0(\mathbb{S}) \cup$$

5. $\neg q \quad res(1, 2).$
6. $\neg r \quad res(1, 3).$
7. $p \vee r \quad res(2, 4).$
8. $p \vee q \quad res(3, 4).$

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

9. $r \quad res(1, 7).$
10. $q \quad res(1, 8).$
11. $p \vee p \quad res(2, 8).$
12. $p \vee p \quad res(3, 7).$
13. $r \quad res(4, 5).$
14. $q \quad res(4, 6).$
15. $p \quad res(5, 8).$
16. $p \quad res(6, 7).$

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

17. $p \quad res(1, 11).$
18. $p \quad res(1, 12).$
19. $\square \quad res(1, 15).$

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

Ejemplo 5.3 Sea \mathbb{S} el conjunto que consta de las siguientes cláusulas:

$$Res_0(\mathbb{S}) = \mathbb{S}$$

1. $r \vee p \vee q$
2. r
3. $\neg q$

$$Res_1(\mathbb{S}) = \mathcal{R}(Res_0(\mathbb{S})) = Res_0(\mathbb{S}) \cup$$

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

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

$$\emptyset$$

Por lo tanto $Res_n(\mathbb{S}) = Res_1(\mathbb{S})$ para toda $n \geq 1$ y como $\Box \notin Res_1(\mathbb{S})$ 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.

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