Lógica Computacional, 2025-2
Unidad 2: Lógica Proposicional

Resolución Binaria

Manuel Soto Romero Luis Fernando Loyola Cruz

11 de marzo de 2025
Facultad de Ciencias UNAM

En esta nota, exploraremos el proceso de transformación de una fórmula lógica a su Forma Normal Negativa (FNN) y su posterior conversión a Forma Normal Conjuntiva (FNC), herramientas fundamentales en lógica computacional y en la automatización de la demostración de teoremas. Comenzaremos definiendo literales y cláusulas, conceptos esenciales en la representación de fórmulas proposicionales. Luego, describiremos cómo obtener la FNN de una fórmula mediante un algoritmo basado en equivalencias lógicas, eliminando condicionales y bicondicionales, aplicando la ley de De Morgan e introduciendo negaciones de manera estructurada. A continuación, convertiremos la FNN a FNC utilizando la distributividad de la disyunción sobre la conjunción.

Más adelante, analizaremos la importancia de la correctud y satisfactibilidad en lógica proposicional, mostrando cómo la FNC facilita la verificación de tautologías y la toma de decisiones en sistemas basados en lógica. Finalmente, presentaremos la regla de resolución binaria de Robinson, un método esencial para la inferencia lógica, junto con ejemplos ilustrativos y su aplicación en algoritmos de saturación, que permiten decidir la insatisfacibilidad de un conjunto de cláusulas.

A lo largo de la nota, veremos cómo estos conceptos son utilizados en problemas de automatización del razonamiento lógico y en aplicaciones dentro de la inteligencia artificial, los lenguajes de programación lógica y la verificación formal.

1. Introducción

Otro método para determinar la corrección de un argumento lógico es la regla de resolución binaria propuesta por John Alan Robinson en 1965. Robinson, un matemático y científico de la computación británico-estadounidense, introdujo este método como una regla de inferencia que revolucionó la demostración automática de teoremas y sentó las bases para el desarrollo de lenguajes de programación lógica como Prolog. Su principal contribución fue la generalización de la regla de inferencia de resolución aplicada a cláusulas, lo que permitió simplificar y sistematizar el proceso de deducción en lógica de primer orden.

Las cláusulas constituyen una clase especial de fórmulas fundamentales en diversas aplicaciones dentro y fuera de la lógica. La forma clausular permite representar cualquier fórmula proposicional como una conjunción de disyunciones, conocidas como cláusulas. A continuación, se presentan algunas definiciones relevantes para el estudio de estos conceptos.

Definición 1. (Literal)

*Se denomina **literal** ℓ a una fórmula atómica o a la negación de una fórmula atómica.*

Definición 2. (Literal Negativa)

*Una literal es **negativa** si corresponde a la negación de una fórmula atómica; en caso contrario, se dice que es **positiva**.*

Definición 3. (Literal Contraria)

*Dada una literal ℓ, definimos su **literal contraria**, denotada por ℓ^c, de la siguiente manera:*

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

*El conjunto $\{\ell, \ell^c\}$ se denomina par de **literales complementarias**.*

Las literales son los elementos fundamentales a partir de las cuales se forman las cláusulas.

Definición 4. (Cláusula)

Se denomina cláusula $\mathcal{C}$ a una literal o a una disyunción de literales.

Para obtener la forma clausular de una fórmula, se aplican transformaciones basadas en equivalencias lógicas, conocidas como formas normales.

2. Formas Normales

Forma Normal Negativa

El objetivo de esta forma normal es obtener una fórmula equivalente en la que no aparezcan condicionales ni bicondicionales y donde, además, los símbolos de negación solo afecten a fórmulas atómicas.

Definición 5. (Forma Normal Negativa)

*Una fórmula φ está en **Forma Normal Negativa** (fnn) si y solo si cumple las siguientes condiciones:*

- ⋆ *φ no contiene bicondicionales ni condicionales.*
- ⋆ *Las negaciones en φ afectan únicamente a fórmulas atómicas.*

Para obtener la fnn de una fórmula, podemos utilizar las equivalencias lógicas estudiadas previamente.

Proposición 1.

Sea φ una fórmula. Es posible encontrar algorítmicamente una fórmula ψ, lógicamente equivalente a φ, tal que ψ esté en Forma Normal Negativa (FNN), también denotada como $fnn(\varphi)$.

Demostración. Dada una fórmula φ, aplicamos el siguiente procedimiento para obtener su Forma Normal Negativa (FNN):

- ⋆ Eliminación de bicondicionales: Sustituimos cada bicondicional $\varphi \leftrightarrow \psi$ utilizando la equivalencia:

$$\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$$

- ⋆ Eliminación de condicionales: Reemplazamos cada condicional $\varphi \rightarrow \psi$ mediante la equivalencia:

$$\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$$

- ⋆ Distribución de negaciones: Aplicamos las leyes de De Morgan para distribuir las negaciones, asegurando que solo afecten a literales.

- ⋆ Eliminación de dobles negaciones: Simplificamos cualquier aparición de $\neg\neg\varphi$ por φ

Finalmente, al aplicar estas transformaciones de manera exhaustiva y adecuada, obtenemos la fórmula $fnn(\varphi)$.

□

Observación 1

La demostración presentada no solo justifica la existencia de la FNN, sino que también describe explícitamente un algoritmo para transformar cualquier fórmula a esta forma. Este enfoque es común en muchos resultados en Ciencias de la Computación, donde una demostración constructiva permite, al mismo tiempo, derivar un procedimiento efectivo para resolver un problema. Este tipo de demostraciones-algoritmo aparecen frecuentemente en teoría de autómatas, compiladores, verificación formal y lógica computacional, donde probar la existencia de una transformación suele implicar definir un método computable para llevarla a cabo.

Ejemplo 1

Encontrar la **Forma Normal Negativa (FNN)** de la siguiente fórmula:

$$(p \leftrightarrow q) \rightarrow \neg(r \rightarrow s)$$

Solución:

- ⋆ Eliminamos bicondicionales:

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

⋆ Eliminamos condicionales:

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

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

⋆ Distribuimos negaciones usando De Morgan:

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

⋆ Eliminamos dobles negaciones:

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

La fórmula obtenida está en **FNN**, ya que no contiene condicionales ni bicondicionales y las negaciones afectan únicamente a literales.

Automatización de la función *fnn*

Siguiendo el algoritmo derivado a partir de la demostración, el objetivo es implementar una función *fnn*:: `LProp -> LProp` que que satisfaga la definición. Para ello, se requieren las siguientes funciones:

```
eliminaBicondicionales :: LProp -> LProp
eliminaCondicionales :: LProp -> LProp
introduceNegaciones :: LProp -> LProp
```

Para eliminar las bicondicionales, podemos usar la regla que aplicamos en el Ejemplo 1:

$$\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$$

Con lo cual obtenemos:

```
eliminaBicondicionales :: LProp -> LProp
eliminaBicondicionales (Var p)    = Var p
eliminaBicondicionales (Cons c)   = Cons c
eliminaBicondicionales (Or p q)   = Or (eliminaBicondicionales p)
                                      (eliminaBicondicionales q)
eliminaBicondicionales (And p q)  = And (eliminaBicondicionales p)
                                       (eliminaBicondicionales q)
eliminaBicondicionales (Impl p q) = Impl (eliminaBicondicionales p)
                                         (eliminaBicondicionales q)
eliminaBicondicionales (Equiv p q) =
    let np = (eliminaBicondicionales p)
        nq = (eliminaBicondicionales q) in
       Conj (Impl np nq) (Impl nq np)
```

Similar a la eliminación de bincondicionales, para eliminar condicionales, lo más adecuado es hacer uso de la siguiente equivalencia:

$$\varphi \to \psi \equiv \neg\varphi \vee \psi$$

Con lo cual tenemos:

```
eliminaCondicionales :: LProp -> LProp
eliminaCondicionales (Var p)   = Var p
eliminaCondicionales (Cte c)   = Cte c
eliminaCondicionales (Or p q)  = Or (eliminaCondicionales p)
                                    (eliminaCondicionales q)
eliminaCondicionales (And p q) = And (eliminaCondicionales p)
                                     (eliminaCondicionales q)
eliminaCondicionales (Impl p q) =
    Or (Neg (eliminaCondicionales p)) (eliminaCondicionales q)
```

Observación 2

La función `eliminaCondicionales` supone que la fórmula que recibe no tiene bicondicionales.

Similar a la eliminación de bicondicionales y condicionales de una fórmula proposicional, para introducir negaciones, lo más adecuado es hacer uso de las siguientes equivalencias

$$\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi \quad \neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi \quad \neg\neg\varphi \equiv \varphi$$

Ejercicio 1

Se deja como ejercicio la implementación de las funciones `introduceNegaciones` y *fnn* que hace uso de todas las funciones definidas.

Ejercicio 2

Obtén la FNN de las siguientes fórmulas:

- $((p \leftrightarrow q) \to (r \vee \neg s)) \leftrightarrow \neg(t \to u)$
- $((p \to q) \leftrightarrow \neg(r \vee s)) \to (t \leftrightarrow \neg u)$
- $\neg((p \leftrightarrow (q \to r)) \vee \neg(s \to t))$

Forma Normal Conjuntiva

La denominada Forma Normal Conjuntiva permite representar cualquier fórmula proposicional como una conjunción de disyunciones, conocidas como cláusulas.

Definición 6. (Forma Normal Conjuntiva)

*Una fórmula proposicional φ está en **Forma Normal Conjuntiva** (fnc) si y sólo si tiene la forma $\mathcal{C}_1 \wedge \cdots \wedge \mathcal{C}_n$ donde cada $\mathcal{C}_i$ es una cláusula.*

En particular, se deduce que cualquier literal y cualquier cláusula están en Forma Normal Conjuntiva.

Proposición 2

Cualquier fórmula proposicional φ puede transformarse mediante un algoritmo en una fórmula ψ tal que $\varphi \equiv \psi$ y ψ esté en Forma Normal Conjuntiva (FNC), también denotada como fnc(φ).

Demostración. Dado que ya hemos demostrado que cualquier fórmula proposicional φ puede transformarse en una fórmula lógicamente equivalente en Forma Normal Negativa (FNN), aplicaremos los siguientes pasos para obtener su Forma Normal Conjuntiva (FNC):

1. Partimos de la FNN de φ. Como φ ya está en FNN, sabemos que:
   - ⋆ No contiene bicondicionales ($\leftrightarrow$) ni condicionales ($\rightarrow$).
   - ⋆ Las negaciones afectan únicamente a literales.
2. Aplicamos la distributividad de la disyunción sobre la conjunción Para transformar la FNN en FNC, utilizamos la equivalencia distributiva:

   $$(\varphi \vee (\psi \wedge \chi)) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \chi)$$

   Procedimiento:
   - ⋆ Si la fórmula en FNN contiene disyunciones dentro de conjunciones, aplicamos esta regla recursivamente hasta obtener una conjunción de disyunciones de literales.
   - ⋆ Este proceso se repite hasta que la fórmula tenga la estructura deseada en FNC.
3. Después de aplicar exhaustivamente la distributividad, obtenemos una fórmula ψ lógicamente equivalente a φ de la forma:

   $$\mathcal{C}_1 \wedge \cdots \wedge \mathcal{C}_n$$

   donde cada $\mathcal{C}_i$ es una disyunción de literales, cumpliendo así la definición de FNC.

El procedimiento descrito es un algoritmo efectivo que transforma cualquier fórmula proposicional en una FNC manteniendo su equivalencia lógica con la fórmula original φ. Por lo tanto, hemos demostrado la proposición.

□

Ejemplo 2

Obtener la Forma Normal Conjuntiva (FNC) de la siguiente fórmula:

$$(p \leftrightarrow (q \vee \neg r)) \rightarrow \neg(s \rightarrow t)$$

Paso 1: Obtener la FNN
1. Eliminamos el bicondicional usando la equivalencia:

$$\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$$

$$(p \rightarrow (q \vee \neg r)) \wedge ((q \vee \neg r) \rightarrow p) \rightarrow \neg(s \rightarrow t)$$

2. Eliminamos los condicionales usando la equivalencia:

$$\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$$

$$(\neg p \vee (q \vee \neg r)) \wedge (\neg(q \vee \neg r) \vee p) \rightarrow \neg(\neg s \vee t)$$

3. Aplicamos la ley de De Morgan para distribuir negaciones:

$$\neg(q \vee \neg r) \equiv \neg q \wedge r, \quad \neg(\neg s \vee t) \equiv s \wedge \neg t$$

Sustituyendo:

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

Paso 2: Distribuir disyunciones sobre conjunciones
Aplicamos la distributividad de la disyunción sobre la conjunción:

$$(\varphi \vee (\psi \wedge \chi)) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \chi)$$

- Distribuyendo en la segunda conjunción:

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

- Expandimos también $s \wedge \neg t$ en la conjunción.
Entonces, obtenemos:

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

Resultado final en FNC:

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

Automatización de la función *fnc*

Siguiendo el algoritmo de la demostración, el objetivo es implementar una función *fnc*`:: LProp -> LProp` que satisfaga la definición. De manera algorítmica puede obtenerse de acuerdo a los siguientes pasos:

1. Obtener la FNN correspondiente de la fórmula.
2. Si la fórmula es una literal, ya está en FNC y simplemente se devuelve como salida.
3. Si la fórmula es una conjunción $\varphi \wedge \psi$ se procesa recursivamente como $fnc(\varphi) \wedge fnc(\psi)$.

4. Si la fórmula des una disyunción $\varphi \vee \psi$ se procesa recursivamente como $fnc(\varphi) \vee fnc(\psi)$ y se transforma la expresión resultante en conjunción mediante las leyes de la distributividad.

$$(\varphi \wedge \psi) \vee (\varphi \wedge \chi) \equiv \varphi \wedge (\psi \vee \chi)$$

Se requiere entonces la implementación de la siguiente función:

```
distribuyeDisyuncion :: LProp -> LProp
```

Para implementar la función `distribuyeDisyuncion` se supone que las fórmulas de entrada φ y ψ son formas conjuntivas.

- ⋆ Si ambas φ y ψ son literales entonces se devuelve $\varphi \vee \psi$.
- ⋆ Si $\varphi = \varphi_1 \wedge \varphi_2$ entonces se aplica la distribuitividad:

$$\varphi \vee \psi \equiv (\varphi_1 \wedge \varphi_2) \vee \psi \equiv (\varphi_1 \vee \psi) \wedge (\varphi_2 \vee \psi)$$

  y se vuelve a aplicar recursivamente la función con las nuevas disyunciones.

- ⋆ Si $\psi = \psi_1 \wedge \psi_2$ la definición es análoga al caso anterior.

Ejercicio 3

Se deja como ejercicio la implementación de las funciones `distribuyeDisyuncion` y *fnc*.

Ejercicio 4

Obtén la FNC de las siguientes fórmulas:

- ⋆ $(p \vee q) \to r$
- ⋆ $(p \vee \neg q) \leftrightarrow (\neg p \vee r)$
- ⋆ $(p \to q) \vee (\neg r \to s)$

Satisfacibilidad mediante FNC

El uso de la forma normal conjuntiva (FNC) simplifica el proceso de determinar si una fórmula dada es una tautología.

Proposición 3

Una cláusula $\mathcal{C} = \ell_1 \vee \cdots \vee \ell_n$ es una tautología ($\models \mathcal{C}$) si y sólo si existe un par de índices i, j tales que $1 \leq i, j \leq n$ y las literales correspondientes son complementarias, es decir $\ell_i^c = l_j$. En otras palabras, una cláusula es siempre verdadera si contiene al menos un par de literales complementarias.

Ejemplo 3

Consideremos la cláusula:

$$\mathcal{C} = (p \vee \neg p \vee q)$$

Aquí, p y $\neg p$ son complementarias, lo que significa que sin importar el valor de p, la cláusula siempre será verdadera. Por lo tanto $\mathcal{C}$ es una tautología.

Ejemplo 3

Consideremos la cláusula:

$$\mathcal{D} = (p \vee q \vee r)$$

En este caso, no hay pares de literales complementarias, por lo que $\mathcal{D}$ no es necesariamente una tautología. Su valor dependerá de la asignación de verdad a p, q y r

La proposición 3 nos brinda un método sencillo para determinar si una fórmula φ es una tautología cuando está en FNC. Supongamos que φ se expresa como la conjunción de cláusulas:

$$\varphi = \mathcal{C}_1 \wedge \cdots \wedge \mathcal{C}_n$$

Podemos seguir los siguientes pasos para determinar si φ es una tautología:

- Para cada cláusula $\mathcal{C}_i$, con $1 \leq i \leq n$, buscamos si contiene un par de literales complementarias.
  - Si todas las cláusulas $\mathcal{C}_i$ contienen al menos un par de literales complementarias, entonces $\vDash \varphi$.
  - Si alguna cláusula no contiene un par de literales complementarias, entonces $\nvDash \varphi$

Ejemplo 4

Consideremos la siguiente fórmula en FNC:

$$\varphi = (p \vee \neg q) \wedge (\neg p \vee q) \wedge (r \vee \neg r)$$

Para determinar si φ es una tautología, analizamos cada cláusula individualmente y buscamos si contiene un par de literales complementarias.

- Primera cláusula: $\mathcal{C}_1 = (p \vee \neg q)$
  - Contiene los literales p y $\neg q$.
  - No hay un par de literales complementarios.
- Segunda cláusula: $\mathcal{C}_2 = (\neg p \vee q)$
  - Contiene los literales $\neg p$ y q.
  - No hay un par de literales complementarios.
- Tercera cláusula: $\mathcal{C}_3 = (r \vee \neg r)$
  - Contiene los literales r y $\neg r$.
  - Como contiene un par de literales complementarios, esta cláusula es una tautología.

Para que φ sea una tautología, todas sus cláusulas deben ser tautologías. Sin embargo, observamos que $\mathcal{C}_1$ y $\mathcal{C}_2$ no contienen pares de literales complementarios, por lo que no son tautologías.
Por lo tanto, $\nvDash \varphi$

Partiendo de estos conceptos, para determinar si una fórmula en FNC es satisfacible, podemos usar el principio de refutación, que establece lo siguiente:

$$\vDash \varphi \text{ si y sólo si } \neg\varphi \text{ es insatisfacible}$$

Esto implica que, en lugar de verificar directamente si φ es satisfacible, podemos analizar su negación $\neg\varphi$. Si $\neg\varphi$ no es una tautología, entonces φ es satisfacible, ya que existe al menos una interpretación donde es verdadera.

Ejemplo 5

Supongamos que tenemos la siguiente fórmula en FNC:

$$\varphi = (p \vee q) \wedge (\neg p \vee r) \wedge (\neg q \vee \neg r)$$

Queremos determinar si φ es satisfacible. Podemos analizarlo directamente buscando un estado para las variables que haga la fórmula verdadera, o bien, aplicar el principio de refutación verificando si $\neg\varphi$ es una tautología.

1. La negación de φ es:
$$\neg\varphi = \neg((p \vee q) \wedge (\neg p \vee r) \wedge (\neg q \vee \neg r))$$
Aplicando De Morgan, obtenemos:
$$\neg\varphi = (\neg(p \vee q) \wedge \neg(\neg p \vee r) \wedge \neg(\neg q \vee \neg r))$$
Aplicamos De Morgan, a cada término:
$$\neg\varphi = (\neg p \wedge \neg q) \vee (p \wedge \neg r) \vee (q \wedge r)$$
Para trabajar con la proposición sobre tautologías en FNC, expresamos esto en FNC:
$$\neg\varphi = (\neg p \vee p) \wedge (\neg p \vee \neg r) \wedge (\neg q \vee p) \wedge (\neg q \vee \neg r) \wedge (q \vee p) \wedge (q \vee r)$$

2. Ahora aplicamos la proposición 3 sobre tautologías en FNC, que nos dice que una cláusula es tautología si contiene un par de literales complementarias. Analicemos cada cláusula.
   - $\star$ $\neg p \vee p$ Contiene $p, \neg p$. Es tautología.
   - $\star$ $\neg p \vee \neg r$ No contiene literales complementarias.

   Podemos continuar verificando todas las cláusulas. Sin embargo, para que $\neg\varphi$ todas sus cláusulas deben ser tautologías, pero observamos que la segunda cláusula no lo es.
   Por lo tanto $\nvDash \neg\varphi$

3. Por el principio de refutación y dado que $\nvDash \neg\varphi$, concluimos que φ es satisfacible.

3. Resolución Binaria con Saturación

En esta sección se introduce la regla de resolución binaria de Robinson junto con un método para determinar la corrección de un argumento lógico.

Definición 7. (Regla de Resolución Binaria Proposicional

Sean $\mathcal{C}_\infty$ y $\mathcal{C}_2$ dos cláusulas y ℓ una literal. La ***regla de resolución binaria proposicional*** *se define de la siguiente manera:*

$$\frac{\mathcal{C}_1 \vee \ell \qquad \mathcal{C}_2 \vee \neg\ell}{\mathcal{C}_1 \vee \mathcal{C}_2}$$

En este caso, decimos que las cláusulas $\mathcal{C}_1$ y $\mathcal{C}_2$ se ***resuelven*** *con respecto a la literal ℓ. La cláusula resultante, $\mathcal{C}_1 \vee \mathcal{C}_2$, de denomina* ***resolvente*** *o* ***resolvente binario****.*

La resolución binaria es una de las reglas fundamentales en la lógica proposicional para la deducción automática. Su importancia radica en que permite simplificar conjuntos de cláusulas al eliminar literales opuestas de dos premisas.

Ejemplo 6

Supongamos que tenemos las siguientes cláusulas:

$$\mathcal{C}_1 = (p \vee r) \quad \mathcal{C}_2 = (\neg p \vee q)$$

Aquí, la literal p aparece en $\mathcal{C}_1$ en forma positiva (p) y en $\mathcal{C}_2$ en forma negativa ($\neg p$). Aplicando la regla de resolución binaria con respecto a p, obtenemos el resolvente:

$$\mathcal{C}_1 \vee \mathcal{C}_2 = (r \vee q)$$

Esto significa que podemos **deducir** la cláusula $r \vee q$ sin necesidad de p.

Lo más importante de este procedimiento es que si las cláusulas originales son verdaderas bajo cierta interpretación, entonces el resolvente también lo será. Esta regla es la base de muchos algoritmos de demostración automática de teoremas y solucionadores SAT, ya que permite deducir la complejidad de un conjunto de cláusulas sin perder información esencial.

Observación 3

Es importante notar que la regla de resolución binaria **no es determinista**, ya que, dado un conjunto de cláusulas, pueden existir múltiples formas de aplicar la resolución dependiendo de la elección de la literal ℓ.

En otras palabras, al resolver un conjunto de cláusulas, podemos elegir diferentes pares de cláusulas que contengan literales complementarias, lo que puede generar distintos resolventes en cada paso. Esta falta de determinismo implica que el orden en el que se apliquen las resoluciones puede afectar el número de pasos necesariso para llegar a una conclusión y, en algunos casos, incluso determinar si se logra una derivación eficiente o no.

Por ejemplo, dado el conjunto de cláusulas:

$$p \vee \neg r \vee s \quad \neg p \vee q \vee r$$

Podemos aplicar la resolución respecto a p, obteniendo el resolvente

$$\neg r \vee s \vee q \vee r$$

O podríamos haber elegido resolver r, lo que daría un camino de deducción diferente.

Esta característica es importante en la implementación de solucionadores SAT y otros sistemas automatizados de razonamiento, donde la estrategia de selección de literales puede influir significativamente en el rendimiento del algoritmo.

Resolución Binaria y Argumentos Correctos

La resolución binaria es un método de decisión en lógica proposicional que emplea el principio de refutación para determinar si una fórmula es consecuencia lógica de un conjunto de premisas. Este método consiste en transformar las fórmulas a FNC y aplicar repetidamente la regla de resolución binaria hasta obtener la cláusula vacía ($\square$) la cual se obtiene al resolver cláusulas como $\neg p$ y p, donde la clausula resultante no contiene literales.

Dado un conjunto de premisas Γ y una fórmula φ, se considera el conjunto de cláusulas $\Gamma \cup \{\neg\varphi\}$. Si al aplicar la resolución obtenemos la cláusula vacía, concluimos que φ es consecuencia lógica de Γ, es decir, $\Gamma \vDash \varphi$. Este procedimiento se conoce como refutación del conjunto de cláusulas, y demustra que $\neg\varphi$ es insatisfacible, lo que implica que φ es correcta en Γ.

Observación 4

La cláusula vacía representa una contradicción absoluta, ya que una disyunción vacía es siempre falsa en cualquier interpretación. En la resolución binaria, obtener la cláusula vacía indica que el conjunto de cláusulas es insatisfacible, lo que permite demostrar la corrección de un argumento mediante refutación.

Ejemplo 7

Demostrar que el siguiente argumento es correcto.

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

1. Pasamos primero que nada, cada fórmula involucrada a FNC.
   a) $p \leftrightarrow (q \vee r)$

$$\begin{array}{lcll} p \leftrightarrow (q \vee r) & \equiv & (p \rightarrow (q \vee r)) \wedge ((q \vee r) \rightarrow p) & \text{Elim. Op.} \\ & \equiv & (\neg p \vee (q \vee r)) \wedge (\neg (q \vee r) \vee p) & \text{Elim. Op.} \\ & \equiv & (\neg p \vee q \vee r) \wedge ((\neg q \wedge \neg r) \vee p) & \text{De Morgan} \\ & \equiv & (\neg p \vee q \vee r) \wedge ((\neg q \vee p) \wedge (\neg r \vee p)) & \text{Distrib.} \\ & \equiv & (\neg p \vee q \vee r) \wedge (\neg q \vee p) \wedge (\neg r \vee p) & \text{Asoc.} \end{array}$$

   b) $p \rightarrow s$

$$p \rightarrow s \quad \equiv \quad \neg p \vee s \quad \text{Elim. Op.}$$

c) q ya está en FNC.
d) s ya está en FNC.

2. Separamos cada cláusula por comas y aplicamos el principio de refutación.

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

3. Aplicamos resolución hasta llegar a la cláusula vacía:

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

$\therefore$ Es un argumento correcto.

Ejemplo 8

Demostrar que el siguiente argumento es correcto.

$$s \leftrightarrow t, t \vee p, s \rightarrow \neg w, w / \therefore p$$

1. Pasamos primero que nada, cada fórmula involucrada a FNC.
a) $s \leftrightarrow t$

$$\begin{array}{rcll} s \leftrightarrow t & \equiv & (s \rightarrow t) \wedge (t \rightarrow s) & \text{Elim. Op.} \\ & \equiv & (\neg s \vee t) \wedge (\neg t \vee s) & \text{Elim. Op.} \end{array}$$

b) $t \vee p$ ya está en FNC.
c) $s \rightarrow \neg w$

$$s \rightarrow \neg w \equiv \neg s \vee \neg w \quad \text{Elim. Op.}$$

d) w ya está en FNC.
e) p ya está en FNC.
f) Separamos cada cláusula por comas y aplicamos el principio de refutación.

$$\{\neg s \vee t, \neg t \vee r, \neg s \vee \neg w, w, \neg p\}$$

2. Aplicamos resolución hasta llegar a la cláusula vacía:

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

∴ Es un argumento correcto.

Algoritmos de Saturación

La resolución binaria es un método esencial en lógica computacional para la automatización de la inferencia lógica. Sin embargo, aplicar la resolución de manera arbitraria o *al tanteo* no es eficiente debido a varias razones fundamentales:

- ⋆ Si las resoluciones se aplican sin estrategia, se pueden generar resolventes innecesarios que aumentan descontroladamente el tamaño del conjunto de cláusulas, dificultando la búsqueda de una solución.
- ⋆ La selección de pares de cláusulas sin un orden definido puede hacer que se exploren combinaciones irrelevantes antes de encontrar la cláusula vacía □, retrasando la detección de la insatisfacibilidad.
- ⋆ Si se aplican resoluciones sin un criterio claro, puede ocurrir que algunas derivaciones esenciales nunca se generen, evitando alcanzar la contradicción.

Para abordar estos problemas, se recurre a algoritmos de saturación, los cuales organizan y sistematizan la aplicación de la regla de resolución para garantizar que todas las combinaciones relevantes se exploren sin redundancias innecesarias.

La saturación en lógica es un procedimiento iterativo que genera todos los resolventes posibles de un conjunto de cláusulas hasta que se cumple un criterio de terminación. Este proceso se formaliza en la n-ésima resolución de un conjunto de cláusulas, definida recursivamente como:

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

donde $\mathcal{R}(\mathbb{S})$ es el conjunto original de cláusulas junto con todos los resolventes posibles generados en cada paso. La saturación garantiza que si un conjunto de cláusulas $\mathbb{S}$ es insatisfacible, la cláusula vacía □ aparecerá en algún $Res_n(\mathbb{S})$ para algún $n \in \mathbb{N}$.

Cuando se implementa un algoritmo de saturación para la resolución binaria, se pueden presentar tres situaciones teóricas:

1. $\square \in Res_n(\mathbb{S})$ para algún n. En este caso, el conjunto de cláusulas $\mathbb{S}$ es insatisfacible, y por lo tanto, el argumento lógico es correcto por refutación.
2. En algún momento, se alcanza un conjunto de cláusulas en el que no se pueden generar más resolventes, es decir, $Res_n(\mathbb{S}) = Res_{n+1}(\mathbb{S})$ y $\square \notin Res_n(\mathbb{S})$. En este caso, el conjunto de cláusulas es satisfacible.
3. Para todo n, se siguen generando nuevas cláusulas sin alcanzar □, lo que indica que el problema es satisfacible, pero la saturación no se detiene.

En la práctica, este tercer caso rara vez ocurre. En cambio, lo que suele suceder es que el proceso de saturación consume demasiados recursos computacionales y se detiene sin haber encontrado □, dejando la satisfacción del conjunto $\mathbb{S}$ indeterminada.

Ejemplo 9

Demostrar que el siguiente argumento es correcto.

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

1. Pasamos primero que nada, cada fórmula involucrada a FNC.
   a) $p \to q$

$$p \to q \quad \equiv \quad \neg p \vee q \quad \text{Elim. Op.}$$

   b) $r \vee s$ ya está en FNC.
   c) $s \to \neg q$

$$s \to \neg q \quad \equiv \quad \neg s \vee \neg q \quad \text{Elim. Op.}$$

   d) $\neg r$ ya está en FNC.
   e) $\neg p$ ya está en FNC.
2. Separamos cada cláusula por comas y aplicamos el principio de refutación.

$$\mathbb{S} = \{\neg p \vee q, r \vee s, \neg s \vee \neg q, \neg r, p\}$$

3. Usando saturación:
   a) $Res_0(\mathbb{S}) = \mathbb{S}$.

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

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

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

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

10.	$\neg p \vee r$	$Res(1,9)$
11.	r	$Res(2,7)$
12.	$\neg q$	$Res(3,8)$
13.	$\neg s$	$Res(3,6)$

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

14.	$\neg p$	$Res(1,12)$

   e) $Res_4(\mathbb{S}) = \mathcal{R}(Res_3(\mathbb{S})) = Res_3(\mathbb{S}) \cup$

15.	$\square$	$Res(5,14)$

$\therefore \square$ pertenece a la tercera resolución de $\mathbb{S}$, así que $\mathbb{S}$ es insatisfacible y por lo tanto el argumento es correcto.

Si bien la saturación permite abordar el problema de manera sistemática, el método puro es computacionalmente costoso. Para mejorar su rendimiento, se emplean estrategias como:

- ⋆ Se descartan resolventes redundantes o innecsarios para evitar un crecimiento incontrolado del conjunto de cláusulas.
- ⋆ Si una cláusula es subsumida por otra más general, se puede eliminar para reducir el tamaño del problema.
- ⋆ Se priorizan ciertas resoluciones, como aquellas que tienen menor cantidad de literales, para acelerar la convergencia hacia la contradicción.

Mientras que la resolución binaria *al tanteo* es poco efectiva debido a la falta de dirección y el crecimiento no controlado del conjunto de cláusulas, el algoritmo de saturación definido en esta sección permite automatizar la deducción lógica de manera estructurada. Este enfoque, combinador con técnicas de optimización constituye la base de muchos sistemas modernos de demostración automática de teoremas y solucionadores SAT como veremos en la próxima nota.

Automatización

En Haskell, este algoritmo puede ser implementado mediante la siguiente función:

```
saturacion :: [Clausula] -> Bool
```

Conclusión

En conclusión, la resolución binaria y los algoritmos de saturación constituyen herramientas fundamentales en la automatización del razonamiento lógico y la verificación formal. Hemos explorado cómo la transformación de fórmulas a Forma Normal Negativa (FNN) y Forma Normal Conjuntiva (FNC) facilita la aplicación sistemática de la regla de resolución, permitiendo la detección de insatisfacibilidad en conjuntos de cláusulas. Además, el principio de refutación proporciona un enfoque práctico para verificar la corrección de argumentos lógicos, garantizando la inferencia mediante la generación estructurada de resolventes.

Si bien la resolución binaria pura puede ser ineficiente debido al crecimiento exponencial del espacio de búsqueda, el uso de algoritmos de saturación optimizados con poda y heurísticas mejora su desempeño en problemas reales. Estas técnicas, combinadas con estrategias modernas de reducción de cláusulas redundantes y selección inteligente de resolventes, forman la base de los solucionadores SAT y otros sistemas de inferencia automática. En futuras notas, exploraremos cómo estas ideas pueden extenderse a lógicas más expresivas y a aplicaciones en inteligencia artificial y verificación de software.

Referencias

[1] Miranda Perea, F. E., Reyes, A. L., González, L. del C., & Linares, S. (2018). *Lógica Computacional: Notas de clase.* Facultad de Ciencias, Universidad Nacional Autónoma de México.

[2] Loyola Cruz, L. F. (2023). *Manual de Prácticas para la Asignatura de Lógica Computacional* (Proyecto de Apoyo a la Docencia). Universidad Nacional Autónoma de México.

[3] Enderton, H. B. (2001). *A Mathematical Introduction to Logic* (2nd ed.). Elsevier.

[**4**] Robinson, J. A. (1965). *A Machine-Oriented Logic Based on the Resolution Principle*. Journal of the ACM, 12(1), 23-41. `https://doi.org/10.1145/321250.321253`

[**5**] Huth, M., & Ryan, M. (2004). *Logic in Computer Science: Modelling and Reasoning about Systems* (2nd ed.). Cambridge University Press.

[**6**] Chang, C.-L., & Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press.

[**7**] Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.