Programación con lógica proposicional

En la Lógica Formal se estudian los principios y métodos a través de los cuales podemos determinar la validez de argumentos, desde el punto de vista solamente de su estructura, sin tomar en cuenta el contenido semántico de las expresiones de los argumentos. De esta manera si se argumenta que:

Todos los Pakistaníes son de Pakistán
Mohammed es Pakistaní
En consecuencia, Mohammed es de Pakistán.

En este argumento, no tomamos en cuenta si los pakistaníes son humanos, perros, pericos o un concepto abstracto de cualquier área.

Tampoco nos importa si Mohammed es un ciudadano de alguna ciudad del mundo o si es el nombre de un perro.

De esta manera desde el punto de vista de su estructura este argumento es válido.

Se hace hincapié que la Lógica no se hace responsable de su aplicación a nivel semántico.

Se puede decir que la Lógica es una herramienta para el análisis de la veracidad de argumentos en base sólo a la estructura de éstos, donde el significado de los elementos que intervienen no es tomado en cuenta.

El argumento anterior tiene dos partes principales:

A) Las premisas:
Todos los Pakistaníes son de Pakistán
Mohammed es Pakistaní
B) La conclusión:
Mohammed es de Pakistán.

De esta manera el argumento es válido, ya que de las premisas sigue la conclusión, lo cual hasta cierto punto nos parece totalmente natural.

Consideremos el siguiente argumento:

Argentina está en África o Argentina está en Asia.
Argentina no está en Asia
En consecuencia, Argentina está en África.

Nuevamente este argumento es válido desde el punto de vista lógico, aún cuando sabemos que la conclusión es falsa.
¿Cómo puede ser esto? ¿A partir de la Lógica se pueden obtener conclusiones equivocadas?

La respuesta es afirmativa, ya que la lógica no verifica el significado de las premisas.
Debido a lo anterior es necesario distinguir entre proposiciones verdaderas y proposiciones lógicamente verdaderas.

Las primeras son verdaderas independientemente de su estructura, mientras que las segundas no lo son. De esta manera, las proposiciones:

Argentina está en África o Argentina está en Asia
Argentina está en África

Son verdaderas lógicamente debido a que la primera es una premisa y a que la segunda ha sido derivada lógicamente de sus premisas.

Las proposiciones son expresiones que pueden ser evaluadas como verdaderas o falsas.

En los lenguajes naturales (Español, Inglés, etc), las proposiciones sólo pueden ser expresiones declarativas y nunca interrogativas o imperativas.

De esta manera las siguientes son proposiciones:

- Hay vida en la tierra
- Una piedra no puede volar
- Todos los hombres de Colombia son solteros
- Chile es la capital del mundo
- Los cantantes no duermen.
- Comer mucho, engorda
- Las montañas cantan bonito
- Los mosquitos viven menos de un año
- El hombre desciende del elefante

Sin embargo, las siguientes no son proposiciones por no poder ser evaluadas como verdaderas ni falsas:

- De veras?
- Teclea exit
- Por favor hagan el trabajo de satisfactibilidad!
- Arriba Argentina!
- ¡Levántate temprano!
- ¿Has entendido lo que es una proposición?
- ¡Estudia esta lección!
- ¿Cuál es la dirección de la página de Lógica Computacional?

En este módulo estudiamos la lógica proposicional, es decir, se estudian los principios para determinar la validez de argumentos conformados con proposiciones. Esto involucra los siguientes tipos de proposiciones:

- Proposiciones simples o átomos
- Proposiciones compuestas

Los átomos o proposiciones simples son tales que no es posible encontrar en ellos otras proposiciones, mientras que las proposiciones compuestas están conformadas de varias proposiciones simples a través de lo que se denomina conectores lógicos, entre los cuales se encuentran: y, o, implica.
Ejemplo de proposiciones compuestas son:

Las montañas cantan bonito o Los mosquitos viven menos de un año.

El hombre desciende del elefante y Comer mucho, engorda.

Conectivas Lógicas

Las conectivas lógicas también se llaman a veces operadores, y son de dos tipos:

Operadores unarios:
NEGACION: not, ¬

Operadores binarios:
CONJUNCION: and, ∧, y
DISYUNCION: or ∨
CONDICIONAL: implies, ⇒, implica
BICONDICIONAL: ⇔

Fórmulas Bien Formadas

El Cálculo Proposicional estudia fórmulas preposicionales simples o compuestas.
Las proposiciones simples o átomos son representadas por símbolos, generalmente las letras del alfabeto A, B, C, ...

Para obtener proposiciones compuestas se utilizan, como se dijo antes, conectores lógicos. Así la proposición compuesta A or B puede corresponder por ejemplo a:

El coronel no tiene quien le escriba
or
La jubilación del Coronel Buendía es insuficiente para su familia

Una fórmula bien formada (fbf) es una expresión que representa una proposición simple o compuesta, la cual esta bien escrita de acuerdo con determinada sintaxis.

Ahora bien, una fbf del Cálculo Proposicional, es una fórmula que está bien escrita de acuerdo con la sintaxis del Cálculo Proposicional.

Las reglas de la sintaxis del Cálculo Proposicional definen, de esta manera, la forma de escribir o reconocer sus fbf's. Estas reglas son:

a) Un átomo es una fórmula bien formada.
b) Si G es una fórmula bien formada entonces ¬G también lo es.
c) Si G y H son fórmulas bien formadas, entonces también lo son:
G ∧ H
G ∨ H
G ⇒ H
G ⇔ H
d) Todas las fbf's se obtienen aplicando a, b y c.

Es necesario puntualizar en la regla c anterior, que es posible utilizar otras conectivas, pero sin embargo son reducibles a las que aquí presentamos.

De esta manera, fijaremos nuestra atención solo a las fbf's que aquí describimos.

Ejemplos de fórmulas bien formadas son:

$P \vee Q$ $P \Rightarrow Q$

Ejemplos de fórmulas que no son bien formadas son: $P \wedge, \Rightarrow Q$.

Interpretación

Asignación de valores de verdad para las proposiciones de una expresión.

El significado de una formula proposicional se puede expresar por medio de una función:
ϖ: prop $\rightarrow$ {verdadero, falso}

La función ϖ es una *función de interpretación* que satisface:

F	**G**	**¬G**	**F ∧ G**	**F ∨ G**	**F ⇒ G**	**F ⇔ G**
V	V	F	V	V	V	V
V	F	F	F	V	F	F
F	V	V	F	V	V	F
F	F	V	F	F	V	V

Si ϖ es una interpretación que asigna a una fórmula dada, el valor de verdad *verdadero*, entonces se dice ser un modelo de F.

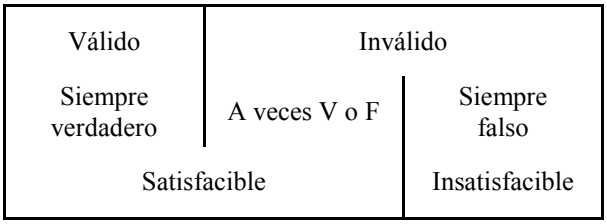
Una fórmula se dice válida si es verdadera bajo cualquier interpretación (tautología).
Una fórmula es inválida si no es válida.
Una fórmula es insatisfascible o inconsistente si es falsa bajo cualquier interpretación (contradicción), sino es satisfascible o consistente.
Que las interprestaciones de una fórmula sean verdaderos o falsos nos encontramos con una contingencia.

F	**¬F**	**F ∨ ¬F**
V	F	V
F	V	V

Válida

F	**¬F**	**F ∧ ¬F**
V	F	F
F	V	F

Inconsistente
(Contradicción)



Dos fórmulas F y G son equivalentes si los valores de verdad de F y G son iguales bajo cualquier interpretación.

Leyes de equivalencia.

$\neg(\neg F) \equiv F$
$F \wedge G \equiv G \wedge F$
$F \vee G \equiv G \vee F$

$(F \wedge G) \wedge H \equiv G \wedge (F \wedge H)$
$(F \vee G) \vee H \equiv G \vee (F \vee H)$

$F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$
$F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$

$F \Leftrightarrow G \equiv F \Rightarrow G \wedge G \Rightarrow F$
$F \Rightarrow G \equiv \neg F \vee G$

$\neg(F \wedge G) \equiv \neg G \vee \neg F$
$\neg(F \vee G) \equiv \neg G \wedge \neg F$

Consecuencias Lógicas

Dadas las fórmulas F1, … , Fn y una fórmula G, se dice que G es una consecuencia lógica (G sigue lógicamente) de F1, … , Fn, si y solo si para cualquier interpretación I, en la cual F1, … , Fn es verdadera, G también lo es.

Pruebas

Método Directo
Dadas las fórmulas F1,F2, … , Fn y una fórmula G, G es una consecuencia lógica de F1,F2, … , Fn, si y solo si:
$(F1 \wedge F2 \wedge \ldots \wedge Fn) \Rightarrow G$ es válida.

Método Indirecto
Dadas las fórmulas F1, F2, … , Fn y una fórmula G, G es una consecuencia lógica de F1, F2, … , Fn, si y solo si:
$(F1 \wedge F2 \wedge \ldots \wedge Fn) \wedge \neg G$ es inconsistente.

Otras definiciones

Literal Proposicional

Es una variable proposicional o la negación de una variable proposicional. Un literal proposicional positivo es simplemente una variable proposicional, un literal proposicional negativo es la negación de una variable proposicional.

Cláusula proposicional

Es una disyunción de literales proposicionales. Es una formula proposicional donde el número de literales proposicionales están conectados con el operador $\vee$.

Ejemplo: $p \vee \neg q \vee r \vee \neg s$

La fórmula proposicional $p \vee \neg(q \wedge r)$ no es una cláusula porque $\neg(q \wedge r)$. Sin embargo la aplicación de las leyes de De Morgan pueden hacer que se vuelva una cláusula proposicional.

Cláusula Proposicional de Horn

Es una cláusula proposicional con un literal positivo como máximo. Una cláusula puede tomar una de las siguientes formas:

1. q
2. $\neg p_1 \vee \ldots \vee \neg p_n \vee q$
3. $\neg p_1 \vee \ldots \vee \neg p_n$

donde $p_1 \ldots p_n$, q son variables proposicionales.

Las cláusulas de la forma (1) y (2) con un literal positivo, se denominan *cláusulas de programa* (o cláusulas definidas). Las cláusulas de Horn de tipo (1) se denominan *cláusulas unidad*.

Las cláusulas de tipo (2), pueden escribirse (usando las leyes de De Morgan) como:

$$\neg(p_1 \wedge \ldots \wedge p_n) \vee q$$

y éstas pueden escribirse como:

$$(p_1 \wedge \ldots \wedge p_n) \Rightarrow q$$

La *cola* de cláusula definida por $(p_1 \wedge \ldots \wedge p_n)$, (vacía en las cláusulas de tipo (1)), y la *cabeza* de la cláusula denotada por q (vacía en las cláusulas de tipo (3)).

Un programa lógico proposicional es un conjunto de cláusulas proposicionales de programa.

Las cláusulas de Horn de tipo (3) son conocidas como *cláusulas meta*.

Resumiendo los conceptos presentados, se introducido la idea de prueba de fórmulas mediante el concepto de "*satisfacción lógica*", que tiene connotaciones semánticas, y que en definitiva es la que interesa cuando se intenta dar un significado a la ejecución de un programa en lógica. Sin embargo no parece evidente encontrar mecanismos automatizables para la prueba, que consideren todas las alternativas posibles: resulta en una explosión combinatoria, tal como se necesita a partir de la definición de satisfacción lógica.

En segundo lugar se presentará el mecanismo de "*inferencia lógica"*, que basado en elementos sintáctico, permite definir derivaciones entre fórmulas, e introducir el concepto de "teoremas", a partir de un conjunto de axiomas y de reglas de inferencias utilizadas. Esta alternativa parece ser más útil desde el punto de vista informático, si se encuentran procedimientos que implementen las reglas del sistema.

Debemos definir entonces algunos conceptos que nos ayudarán a encontrar un mecanismo automatizable.

Completitud

- Sea un programa lógico **P**, y una cláusula **p**, que corresponde a una invocación del programa **P**.
- Sea **Q** una regla de inferencia.

- Se dice que **Q** es *completa* si se cumple que:
  - p es deducible lógicamente de P, utilizando Q sii p es consecuencia lógica de P.

Reglas de Inferencias

La inferencia lógica es un mecanismo de derivación sintáctica que a partir de un conjunto dado de fórmulas permite derivar nuevas fórmulas, utilizando operaciones que se denominan **reglas de inferencia**.

El conjunto inicial de formulas son sentencias que se denominan axiomas. Los axiomas junto con las reglas de inferencias constituyen un sistema formal.

Permiten la deducción de nuevas proposiciones a partir de otras dadas. Así se relaciona el hecho de que una nueva proposición sea verdadera, a partir de la veracidad de las proposiciones originales

Reglas de deducción clásicas.

Modus Ponens	Modus Tollens	Silogismo Disyuntivo
A $A \Rightarrow B$ B	$\neg B$ $A \Rightarrow B$ $\neg A$	$P \vee Q$ $\neg P$ Q

Resolución proposicional

Es una regla de inferencia que se utiliza en programación lógica para realizar deducciones.

Supongamos que tenemos $C_1 \vee p$ y $C_2 \vee \neg p$, donde C_1 y C_2 son cláusulas y p es una variable proposicional.

La regla de resolución permite obtener la expresión $C_1 \vee C_2$ que no tiene en cuenta la variable p. Podemos definir resolución como la siguiente regla de inferencia:

$C_1 \vee p,\ C_2 \vee \neg p \ \vdash_{Res}\ C_1 \vee C_2$

Donde el símbolo $\vdash_{Res}$, denota deducción usando la regla de resolución.

Tomemos por ejemplo una cláusula C_1 de tipo (3) y C_2 de tipo (2):

$C_1 =_T \neg q_1 \vee ... \vee \neg q_n$
$C_2 =_T \neg r_1 \vee ... \vee \neg r_m \vee s$

donde $q_1 \ldots q_n$, $r_1 \ldots r_n$ don variables proposicionales y n y m son enteros no negativos.

podemos tener dos cláusulas:

a) $C_1 \vee p$
b) $C_2 \vee \neg p$

podemos escribirlas como:

a) $(q_1 \wedge ... \wedge q_n) \Rightarrow p$
b) $(r_1 \wedge ... \wedge r_m \wedge p) \Rightarrow s$

podemos expresar la disyunción (sin la variable p) de éstas como:

$$C_1 \vee C_2 \quad =_T \ \neg q_1 \vee ... \vee \neg q_n \vee \neg r_1 \neg \vee ... \vee \neg r_m \vee s$$
$$=_T \ (q_1 \wedge ... \wedge q_n \wedge r_1 \wedge ... \wedge r_m) \Rightarrow s$$

Resolución puede verse, en términos de cláusulas de Horn como:

$(q_1 \wedge ... \wedge q_n) \Rightarrow p,\ \ (r_1 \wedge ... \wedge r_m \wedge p) \Rightarrow s \quad \vdash_{Res} \quad (q_1 \wedge ... \wedge q_n \wedge r_1 \wedge ... \wedge r_m) \Rightarrow s$

Esto puede entenderse como: la cláusula (b) dice que para derivar s necesitamos derivar r_1,... r_m, p. Sin embargo para derivar p solo necesitamos derivar $q_1, \ldots q_n$.

Refutación (Contradicción) y Deducción

Resolución es una regla de inferencia que se utiliza en la programación lógica para realizar deducciones. Un programa lógico consiste en un conjunto de cláusulas de programa, que son consideradas como un conjunto de hipótesis. La regla de resolución puede aplicarse a la hipótesis para deducir consecuentes.

Si podemos inferir a partir de una cláusula de programa que una proposición **p** es verdadera, podemos decir que **p** tiene éxito. Si no podemos decir que **p** es verdadero, podemos decir que **p** falla. Esto no significa que **p** es falso; significa simplemente que **p** no puede inferirse como verdadero de las cláusulas del programa dado.

Sin embargo, hay un método particular usado en programación lógica para realizar estas deducciones denominado *refutación*.

Supongamos que tenemos un programa lógico al cual se le presentan *queries* (consultas = conjunción de variables proposicionales)

La pregunta que se puede hacer al programa es:

Si las cláusulas de programa se toman como hipótesis, puede la consulta deducirse de estas hipótesis usando la regla de resolución?

ó

Dado un programa lógico P (que es un conjunto de cláusulas de programa) y una consulta Q, puede establecerse la deducción:

$P \vdash_{Res} Q$*?*

Teorema
Sea Q una fórmula y F un conjunto de fórmulas. Se dice que Q es ***deducible lógicamente*** *a partir de F, sii el conjunto formado por F y ¬Q es* ***inconsistente****.*

La única regla que usaremos es la regla de resolución mediante el método de *refutación o contradicción*.

Agregaremos ¬Q como una hipótesis y usaremos resolución para establecer una contradicción en la forma de cláusula vacía. Usando resolución, la única manera de obtener la cláusula vacía es aplicando la regla a la variable proposicional p y ¬p, en la cual tenemos una contradicción. La cláusula vacía es denotado por □.

Una consulta (query) es una conjunción de variables proposicionales y tiene la siguiente forma:

$q_1 \wedge ... \wedge q_n$

cuya negación es requerida para formar una cláusula meta (Cláusula de Horn de tipo (3)).

$\neg q_1 \vee \ldots \vee \neg q_n$

Esta restricción es debido a que trabajaremos solo con cláusulas de Horn, con una cierta cantidad de cláusulas de programa y una única cláusula meta. En esta definición, un literal negativo no puede ser un query (la negación tiene un tratamiento especial).

Ejemplo 1: Supongamos que tenemos el siguiente programa:

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

al que se le presenta la consulta r. Agregamos la hipótesis adicional ¬r y probamos:

1.	**¬p ∨ ¬q ∨ r**	
2.	**p**	Hipótesis
3.	**q**	
4.	**¬r**	Hipótesis agregada
5.	**¬q ∨ r**	Res por 1 y 2
6.	**r**	Res por 3 y 5
7.	□	Res por 4 y 6

Se dedujo la cláusula vacía por lo tanto las hipótesis combinadas con **¬r** no puede satisfacerse, por lo tanto la consulta r es exitosa.

Ejemplo 2: Supongamos que tenemos el siguiente programa:

a) $(p \wedge q \wedge r) \Rightarrow s$
b) $(t \wedge w) \Rightarrow r$
c) q
d) $(v \wedge r) \Rightarrow p$
e) t
f) v
g) $v \Rightarrow w$

o lo que es lo mismo:

1. $\neg p \vee \neg q \vee \neg r \vee s$
2. $\neg t \vee \neg w \vee r$
3. q
4. $\neg v \vee \neg r \vee p$
5. t
6. v
7. $\neg v \vee w$

aplicamos la consulta s, lo que significa que ¬s es agregado como hipótesis extra.

1.	**¬p ∨ ¬q ∨ ¬r ∨ s**	
2.	**¬t ∨ ¬w ∨ r**	
3.	**q**	
4.	**¬v ∨ ¬r ∨ p**	Hipótesis
5.	**t**	
6.	**v**	
7.	**¬v ∨ w**	
8.	**¬s**	Hipótesis agregada
9.	**¬p ∨ ¬q ∨ ¬r**	Res por 1 y 8
10.	**¬p ∨ ¬q ∨ ¬t ∨ ¬w**	Res por 2 y 9
11.	**¬p ∨ ¬t ∨ ¬w**	Res por 3 y 10
12.	**¬t ∨ ¬w ∨ ¬v ∨ ¬r**	Res por 4 y 11
13.	**¬w ∨ ¬v ∨ ¬r**	Res por 5 y 12
14.	**¬w ∨ ¬r**	Res por 6 y 13
15.	**¬v ∨ ¬r**	Res por 7 y 14
16.	**¬t ∨ ¬w ∨ ¬v**	Res por 2 y 15
17.	**¬w ∨ ¬v**	Res por 5 y 16
18.	**¬v**	Res por 7 y 17

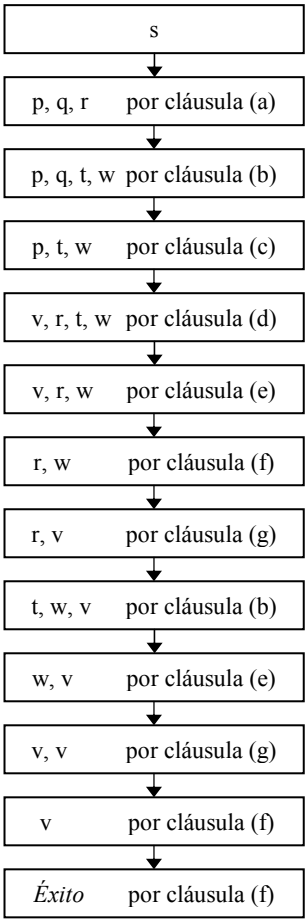
19. □ Res por 6 y 18

Árbol de resolución

Los mismos argumentos pueden presentarse en la forma de *árbol de resolución*. En este caso consideraremos el programa clausal escrito en la forma de conjunciones e implicaciones.

Cada literal positivo de la consulta se considera como una submeta, las cuales son derivadas en turno. Veremos que cada submeta coincide con la cabeza de una (o mas) cláusulas de programa. Así, la cola de cada cláusula elegida nos da mas submetas para derivar (o ninguna si es una cláusula unidad). Este proceso se continúa hasta que todas las submetas son derivadas (o hasta que todas las posibilidades fueron probadas y las submetas no pueden ser derivadas).

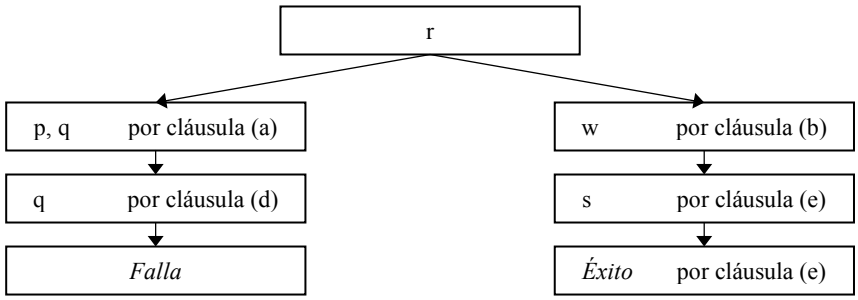
El árbol de resolución (de una única rama) para el ejemplo anterior sería:



Ejemplo 3: supongamos el siguiente programa donde hay que considerar dos ramas.

a) $(p \wedge q) \Rightarrow r$
b) $w \Rightarrow r$
c) $s \Rightarrow w$
d) p
e) s

ahora que se presenta la consulta r. De nuevo agregamos ¬r como hipótesis agregada. En esta oportunidad existen dos cláusulas de programa que pueden elegirse (a) o (b), por lo tanto debemos crear dos ramas en el árbol como se muestra en la figura:



Note que la primer rama falla. cuando esto ocurre, debemos retornar al punto en la deducción donde hay una cláusula alternativa y empezar nuevamente con la otra rama (backtracking).

Mediante resolución debemos considerar las dos ramas:

Rama izquierda		**Rama derecha**	
1. **¬p ∨ ¬q ∨ r**		**1.** **¬p ∨ ¬q ∨ r**	
2. **¬w ∨ r**		**2.** **¬w ∨ r**	
3. **¬s ∨ w**	Hipótesis	**3.** **¬s ∨ w**	Hipótesis
4. **p**		**4.** **p**	
5. **s**		**5.** **s**	
6. **¬r**	Hipótesis agregada	**6.** **¬r**	Hipótesis agregada
7. **¬p ∨ ¬q**	Res por 1 y 6	**7.** **¬w**	Res por 1 y 6
8. **¬q**	Res por 4 y 7	**8.** **¬q**	Res por 4 y 7
falla		**9.** □	Res por 5 y 8

En la rama de la izquierda, no hay una cláusula que contenga q para resolver con ¬q, por lo tanto la derivación falla.

Negación en programación lógica

Como vimos, los programas lógicos solo expresan información positiva. No es posible mostrar que un literal negativo es consecuencia de un programa lógico.

La técnica mayormente usada en programación lógica es considerar cualquier consulta como falsa si no puede derivarse del programa. Esto se conoce con el nombre de *Negación por falla* o por *mundo cerrado*. Por supuesto, desde el punto de vista estricto, la técnica del mundo cerrado no es ideal, debido a que una sentencia que no fue considerada, puede no necesariamente ser falsa aunque la técnica la considere falsa.

Sin embargo esta es la técnica que usa Prolog.

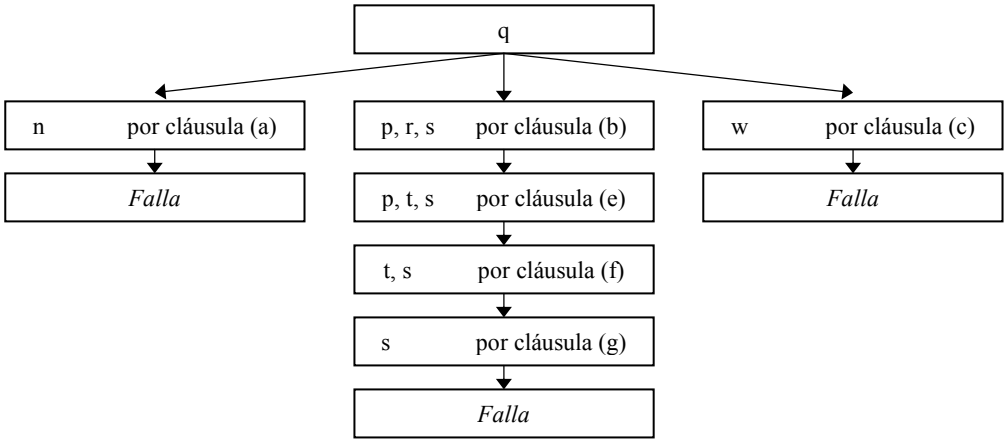
La regla puede expresarse como:

"Si P es un programa proposicional, q es una consulta y P |– q *no puede establecerse, podemos deducir que* P |– ¬q*."*

Ejemplo 4: Consideremos el siguiente programa proposicional:

a) $n \Rightarrow q$
b) $(p \wedge r \wedge s) \Rightarrow q$
c) $w \Rightarrow q$
d) $u \Rightarrow n$
e) $t \Rightarrow r$
f) p
g) t

Para mostrar que $\neg q$ puede derivarse del programa, lo presentamos primero con la consulta q y consideramos todos los caminos del árbol de resolución.



Como se ve en el árbol, q no puede derivarse del programa, por lo tanto, por aplicación de mundo cerrado, podemos decir que $\neg q$ se deriva del programa.

Resolución SLD

Como vimos, en la mayoría de los árboles de resolución, hay un punto en donde se hace necesario hacer una elección de la cláusula a utilizar. El método que usaremos es el *top-down*, esto signifique que la cláusulas se utilizarán en el orden en que fueron escritas desde arriba hacia abajo. Esto es lo mismo que trabajar en *primero en profundidad* de izquierda a derecha en el árbol de resolución.

En los ejemplos anteriores consideramos consultas en donde se involucraban solo una variable proposicional. Sin embargo, las consultas pueden se conjunciones de varias variables. Por lo tanto puede pasar que más de una variable pueda emparejarse con la cabeza de más de una cláusula. En la automatización del proceso de deducción, se hace necesario introducir una función de selección que elija que variable considerar de la cláusula meta.

El uso de la función de selección con resolución se denomina Resolución SLD. (**S**elective **L**inear for **D**efinite clauses).

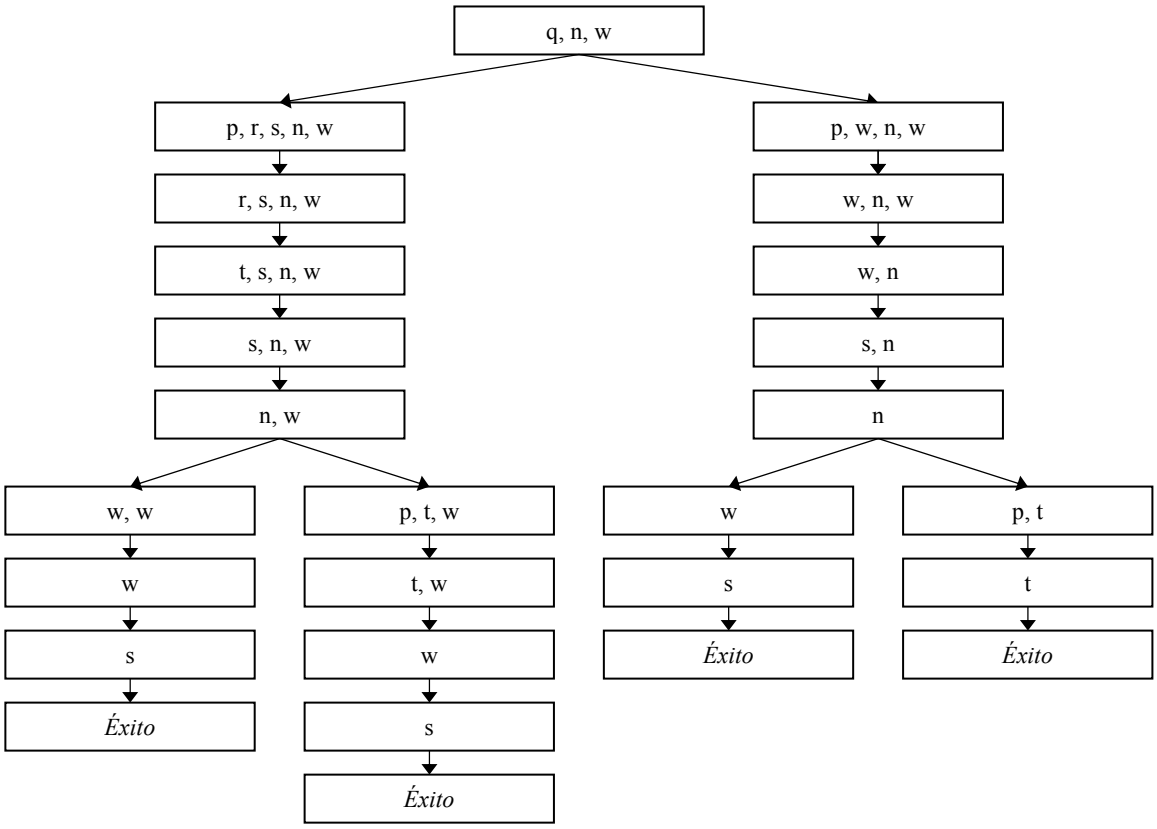
Prolog utiliza la función que elige la meta de más a la izquierda.

Ejemplo 5: Consideremos el siguiente programa:

a) $(p \wedge r \wedge s) \Rightarrow q$
b) $(p \wedge w) \Rightarrow q$

c) $w \Rightarrow n$
d) $(p \wedge t) \Rightarrow n$
e) $s \Rightarrow w$
f) $t \Rightarrow r$
g) p
h) t
i) s

Ahora la consulta es: $q \wedge n \wedge w$. Tenemos el siguiente árbol de Resolución SLD.



En este ejemplo se utilizó Resolución SLD, en el cual se eligió siempre la variable de más a la izquierda. Esta es la función de selección que se eligió, aunque podría usarse siempre la variable de más a la derecha. Note que en este caso, todas las ramas del árbol son exitosas, aunque no es necesario continuar con la derivación por todas las ramas si se puede derivar exitosamente por alguna de ellas.