Lógica Computacional
Proyecto 1: Implementación de la solución de un problema lógico

Rubí Rojas Tania Michelle
Universidad Nacional Autónoma de México
taniarubi@ciencias.unam.mx
de cuenta: 315121719

24 de marzo de 2019

La parte teórica del proyecto está basado en las notas de clase del curso de Lógica Computacional impartido por la profesora Estefanía Prieto Larios y se nutrió con las sugerencias y correcciones de la misma. El propósito de este proyecto es realizar la implementación computacional de la solución a un clásico problema lógico.

1. Lógica Proposicional

1.1. Definición

La lógica proposicional es el sistema lógico más simple. Se encarga del manejo de proposiciones mediante conectivos lógicos. Una proposición es un enunciado que puede calificarse de verdadero o falso.

Ejemplo 1. *Enunciados que son proposiciones.*

- *Los números pares son divisibles por dos.*
- *Una ballena no es roja.*
- *He pasado mis vacaciones en Grecia.*

Ejemplo 2. *Enunciados que no son proposiciones.*

- *¡Auxilio, me desmayo!*
- *No sé si vendrán al viaje.*
- $x + y$

1.2. Sintaxis de la lógica proposicional

Definimos ahora un lenguaje formal para la lógica proposicional.
El alfabeto consta de:

- Símbolos o variables proposicionales (un número infinito) : $p_1, ..., p_n, ...$
- Constantes lógicas: $\bot, \top$
- Conectivos u operadores lógicos: $\neg, \wedge, \vee \rightarrow, \leftrightarrow$
- Símbolos auxiliares: $(,)$

El conjunto de expresiones o fórmulas atómicas, denotado $ATOM$ consta de:

- Las variables proposicionales: $p_1, ..., p_n, ...$
- Las constantes $\bot, \top$

Las expresiones que formarán nuestro lenguaje $PROP$, llamadas usualmente fórmulas, se definen recursivamente como sigue:

- Si $\varphi \in ATOM$ entonces $\varphi \in PROP$. Es decir, toda fórmula atómica es una fórmula.
- Si $\varphi \in PROP$ entonces $(\neg\varphi) \in PROP$.
- φ, ψ entonces $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi), (\varphi \leftrightarrow \psi) \in PROP$.
- Son todas.

Ejemplo 3. *Representación de enunciados en Lógica Proposicional.*

i) El enunciado ***nuestra bandera es blanca y celeste*** *se puede ver en lógica proposicional como*

$$p \wedge q$$

donde p = nuestra bandera es blanca, y q = nuestra bandera es celeste.

ii) El enunciado ***está nublado por lo que va a llover; entonces no saldremos*** *se puede ver en lógica proposicional como*

$$(a \rightarrow b) \rightarrow \neg c$$

donde a = está nublado, b = va a llover y c = saldremos.

1.3. Semántica de la lógica proposicional

Definición 1. *El tipo de valores booleanos denotado* ***Bool*** *se define como* $Bool = \{0, 1\}$

Definición 2. *Un estado o asignación de las variables (proposicionales) es una función*

$$\mathcal{I} : (VarP) \to Bool$$

Dadas n variables proposicionales existen 2^n *estados distintos para estas variables.*

Definición 3. *Dado un estado de las variables* $\mathcal{I} : (VarP) \to Bool$*, definamos la interpretación de las fórmulas con respecto a* $\mathcal{I}$ *como la función* $\mathcal{I}^\star : PROP \to Bool$ *tal que:*

- $\mathcal{I}^\star(p) = \mathcal{I}(p)$ *para* $p \in VarP$*, es decir,* $\mathcal{I}^\star|_{VarP} = \mathcal{I}$
- $\mathcal{I}^\star(\top) = 1$
- $\mathcal{I}^\star(\bot) = 0$
- $\mathcal{I}^\star(\neg\varphi) = 1$ *sii* $\mathcal{I}^\star(\varphi) = 0$
- $\mathcal{I}^\star(\varphi \wedge \psi) = 1$ *sii* $\mathcal{I}^\star(\varphi) = \mathcal{I}^\star(\psi) = 1$
- $\mathcal{I}^\star(\varphi \vee \psi) = 0$ *sii* $\mathcal{I}^\star(\varphi) = \mathcal{I}^\star(\psi) = 0$
- $\mathcal{I}^\star(\varphi \to \psi) = 0$ *sii* $\mathcal{I}^\star(\varphi) = 1$ *e* $\mathcal{I}^\star(\psi) = 0$
- $\mathcal{I}^\star(\varphi \leftrightarrow \psi) = 1$ *sii* $\mathcal{I}^\star(\varphi) = \mathcal{I}^\star(\psi)$

Notemos que dado un estado de las variables $\mathcal{I}$, la interpretación $\mathcal{I}^\star$ generada por I está determinada de manera única, por lo que de ahora en adelante escribiremos simplemente $\mathcal{I}$ en lugar de $\mathcal{I}^\star$.

Ejemplo 4. *Interpretación de fórmulas con valores arbitrarios.*

i) $(s \vee t) \leftrightarrow (s \wedge t)$
Si $\mathcal{I}(s) = 1$, $\mathcal{I}(t) = 0$*, por la definición de* $\mathcal{I}$ *tenemos que* $\mathcal{I}(s \vee t) = 1$ *y* $\mathcal{I}(s \wedge t) = 0$*, por lo que*

$$\mathcal{I}((s \vee t \leftrightarrow (s \wedge t))) = 0$$

ii) $(p \wedge q) \to \neg(r \wedge q)$
Si $\mathcal{I}(p) = 1$, $\mathcal{I}(q) = 1$, $\mathcal{I}(r) = 0$*, por la definición de* $\mathcal{I}$ *tenemos que* $\mathcal{I}(p \wedge q) = 1$, $\mathcal{I}(r \wedge q) = 0$ *y* $\mathcal{I}(\neg(r \wedge q)) = 1$*, por lo que*

$$\mathcal{I}((p \wedge q) \to \neg(r \wedge q)) = 1$$

El siguiente lema es de importancia para restringir las interpretaciones de interés al analizar una fórmula.

Lema 1 (**Coincidencia**). *Sean $\mathcal{I}_1, \mathcal{I}_2 : PROP \to Bool$ dos estados que coinciden en las variables proposicionales de la fórmula φ, es decir, $\mathcal{I}_1(p) = \mathcal{I}_2(p)$ para toda $p \in vars(\varphi)$. Entonces $\mathcal{I}_1(\varphi) = \mathcal{I}_2(\varphi)$*

Demostración. Inducción estructural sobre φ.

`Base de inducción.` φ es atómica.

- $\varphi = \top$. Notemos que $\mathcal{I}_1(\top) = \mathcal{I}_2(\top) = 1$ siempre.
- $\varphi = \bot$. Notemos que $\mathcal{I}_1(\bot) = \mathcal{I}_2(\bot) = 0$ siempre.
- $\varphi = VarP$. Como la única fórmula atómica que ocurre en φ es φ, tenemos que $\mathcal{I}_1(\varphi) = \mathcal{I}_2(\varphi)$.

`Hipótesis de inducción.` Supongamos que el argumento es cierto para la fórmula φ', es decir, que se cumple $\mathcal{I}_1(\varphi') = \mathcal{I}_2(\varphi')$.

`Paso inductivo.` Tenemos 5 casos:

- $\varphi = \neg\varphi'$. Supongamos que $\mathcal{I}_1 = \mathcal{I}_2$ coinciden en las variables proposicionales de $\neg\varphi'$; pero las variables de $\neg\varphi$ son las mismas variables de φ, y luego por hipótesis de inducción tenemos que $\mathcal{I}_1(\varphi) = \mathcal{I}_2(\varphi)$. Así,

$$\begin{aligned} \mathcal{I}_1(\neg\varphi) = 1 &\Leftrightarrow \mathcal{I}_1(\varphi) = 0 && \text{por definición de } \mathcal{I} \\ &\Leftrightarrow \mathcal{I}_2(\varphi) = 0 && \text{por Hipótesis de Inducción} \\ &\Leftrightarrow \mathcal{I}_2(\neg\varphi) = 1 && \text{por definición de } \mathcal{I} \end{aligned}$$

  Por lo tanto, $\mathcal{I}_1(\varphi) = \mathcal{I}_2(\varphi)$.

- $\varphi = \phi \wedge \psi$. Entonces tenemos que

$$\begin{aligned} \mathcal{I}_1(\phi \wedge \psi) = 1 &\Leftrightarrow \mathcal{I}_1(\phi) = \mathcal{I}_1(\psi) = 1 && \text{por definición de } \mathcal{I} \\ &\Leftrightarrow \mathcal{I}_2(\phi) = \mathcal{I}_2(\psi) = 1 && \text{por Hipótesis de Inducción} \\ &\Leftrightarrow \mathcal{I}_2(\phi \wedge \psi) = 1 && \text{por definición de } \mathcal{I} \end{aligned}$$

  Por lo tanto, $\mathcal{I}_1(\varphi) = \mathcal{I}_2(\varphi)$.

- $\varphi = \phi \vee \psi$. Entonces tenemos que

$$\begin{aligned} \mathcal{I}_1(\phi \vee \psi) = 0 &\Leftrightarrow \mathcal{I}_1(\phi) = \mathcal{I}_1(\psi) = 0 && \text{por definición de } \mathcal{I} \\ &\Leftrightarrow \mathcal{I}_2(\phi) = \mathcal{I}_2(\psi) = 0 && \text{por Hipótesis de Inducción} \\ &\Leftrightarrow \mathcal{I}_2(\phi \vee \psi) = 0 && \text{por definición de } \mathcal{I} \end{aligned}$$

  Por lo tanto, $\mathcal{I}_1(\varphi) = \mathcal{I}_2(\varphi)$.

- $\varphi = \phi \to \psi$. Entonces tenemos que

$$\begin{aligned} \mathcal{I}_1(\phi \to \psi) = 0 &\Leftrightarrow \mathcal{I}_1(\phi) = 1, \mathcal{I}_1(\psi) = 0 && \text{por definición de } \mathcal{I} \\ &\Leftrightarrow \mathcal{I}_2(\phi) = 1, \mathcal{I}_2(\psi) = 0 && \text{por Hipótesis de Inducción} \\ &\Leftrightarrow \mathcal{I}_2(\phi \to \psi) = 0 \end{aligned}$$

  Por lo tanto, $\mathcal{I}_1(\varphi) = \mathcal{I}_2(\varphi)$.

- $\varphi = \phi \leftrightarrow \psi$. Entonces tenemos que

$$\begin{aligned} \mathcal{I}_1(\phi \leftrightarrow \psi) = 1 &\Leftrightarrow \mathcal{I}_1(\phi) = \mathcal{I}_1(\psi) && \text{por definición de } \mathcal{I} \\ &\Leftrightarrow \mathcal{I}_2(\phi) = \mathcal{I}_2(\psi) && \text{por Hipótesis de Inducción} \\ &\Leftrightarrow \mathcal{I}_2(\phi \leftrightarrow \psi) = 1 && \text{por definición de } \mathcal{I} \end{aligned}$$

  Por lo tanto, $\mathcal{I}_1(\varphi) = \mathcal{I}_2(\varphi)$.

□

El lema anterior implica que aún cuando existen una infinidad de estados, dada una fórmula φ basta considerar únicamente aquellos que difieren en las variables proposicionales de φ, a saber 2^n estados distintos si φ tiene n variables proposicionales.

1.3.1. Conceptos semánticos básicos

Definición 4. *Sea φ una fórmula. Entonces*

- *Si $\mathcal{I}(\varphi) = 1$ para toda interpretación $\mathcal{I}$ decimos que φ es una tautología o una fórmula válida y escribimos $\models \varphi$.*
- *Si $\mathcal{I}(\varphi) = 1$ para alguna interpretación $\mathcal{I}$ decimos que φ es satisfacible o que $\mathcal{I}$ es modelo de φ y escribimos $\mathcal{I} \models \varphi$.*
- *Si $\mathcal{I}(\varphi) = 0$ para alguna interpretación $\mathcal{I}$ decimos que φ es falsa o insatisfacible en $\mathcal{I}$ o que $\mathcal{I}$ no es modelo de φ y escribimos $\mathcal{I} \not\models \varphi$.*
- *Si $\mathcal{I}(\varphi) = 0$ para toda interpretación $\mathcal{I}$ decimos que φ es una contradicción o fórmula no satisfacible.*

Similarmente, si Γ es un conjunto de fórmulas decimos que:

- Γ es satisfacible si tiene un modelo, es decir, si existe una interpretación $\mathcal{I}$ tal que $\mathcal{I}(\varphi) = 1$ para toda $\varphi \in \Gamma$.
- Γ es insatisfacible si no tiene un modelo, es decir, si no existe una interpretación $\mathcal{I}$ tal que $\mathcal{I}(\varphi) = 1$ para toda $\varphi \in \Gamma$.

Ejemplo 5. *Sean φ una fórmula y Γ un conjunto de fórmulas.*

i) $\varphi = p \vee \neg p$
Notemos que si $\mathcal{I}(p) = 1$ entonces $\mathcal{I}(\varphi) = 1$ pero si $\mathcal{I}(p) = 0$ entonces también $\mathcal{I}(\varphi) = 1$. Como en ambos casos obtenemos que $\mathcal{I}(\varphi) = 1$, entonces φ es una tautología.

ii) $\Gamma = \{p \to q, r \to s, \neg s\}$
Si $\mathcal{I}(s) = \mathcal{I}(r) = \mathcal{I}(p) = 0$, entonces $\mathcal{I}(\Gamma) = 1$, por lo que Γ es satisfacible en $\mathcal{I}$.

iii) $\varphi = p \to (q \vee r)$
Si $\mathcal{I}(p) = 1$ y $\mathcal{I}(q) = \mathcal{I}(r) = 0$, entonces $\mathcal{I}(\varphi) = 0$, por lo que φ es insatisfacible en $\mathcal{I}$

iv) $\Gamma = \{p \to q, \neg(q \vee s), s \vee p\}$
Notemos que Γ *es insatisfacible, pues supóngase que existe una interpretación* $\mathcal{I}$ *tal que* $\mathcal{I}(\Gamma) = 1$. *Entonces se tiene que* $\mathcal{I}(\neg(q \vee s)) = 1$, *por lo que* $\mathcal{I}(\neg q) = \mathcal{I}(\neg s) = 1$. *Además, como* $\mathcal{I}(p \to q) = 1$, *entonces* $\mathcal{I}(p) = 0$, *puesto que el consecuente de la implicación es falso. De esto último se tiene que* $\mathcal{I}(s) = 1$, *dado que* $\mathcal{I}(s \vee p) = 1$. *De manera que se tiene* $\mathcal{I}(\neg s) = 1 = \mathcal{I}(s)$, *lo cual es imposible. Por lo tanto, no puede existir una interpretación* $\mathcal{I}$ *que satisfaga a* Γ.

Proposición 1. *Sea* $\Gamma = \{\varphi_1, ..., \varphi_n\}$ *un conjunto de fórmulas.*

- Γ *es satisfacible si y sólo si* $\varphi_1 \wedge ... \wedge \varphi_n$ *es satisfacible.*
- Γ *es insatisfacible si y sólo si* $\varphi_1 \wedge ... \wedge \varphi_n$ *es insatisfacible.*

2. Consecuencia Lógica

Definición 5. *Sean* Γ *un conjunto de fórmulas y* φ *una fórmula. Decimos que* φ *es consecuencia lógica de* Γ *si para toda interpretación* $\mathcal{I}$ *que satisface a* Γ *se tiene que* $\mathcal{I}(\varphi) = 1$. *Es decir, si se cumple que siempre que* $\mathcal{I}$ *satisface a* Γ *entonces necesariamente* $\mathcal{I}$ *satisface a* φ. *En tal caso escribimos* $\Gamma \models \varphi$.

Nótese que la relación de consecuencia lógica está dada por una implicación de la forma

$$\mathcal{I}(\Gamma) = 1 \Rightarrow \mathcal{I}(\varphi) = 1$$

lo cual informalmente significa que *todo modelo de* Γ *es modelo de* φ.
Obsérvese la sobrecarga del símbolo $\models$ que previamente utilizamos para denotar satisfacibilidad $\mathcal{I} \models \varphi$ y tautologías $\models \varphi$.

Ejemplo 6. *Considérese el siguiente conjunto* $\Gamma = \{q \to p, p \leftrightarrow t, t \to s, s \to r\}$. *Muestre que* $\Gamma \models q \to r$.
Solución: Sea $\mathcal{I}$ *un modelo de* Γ. *Tenemos que demostrar que* $\mathcal{I}(q \to r) = 1$. *Si* $\mathcal{I}(q) = 0$ *entonces* $\mathcal{I}(q \to r) = 1$ *y terminamos. En otro caso se tiene que* $\mathcal{I}(q) = 1$ *de donde* $\mathcal{I}(p) = 1$ *pues* $\mathcal{I}(q \to p) = 1$. *Entonces se tiene que* $\mathcal{I}(t) = 1$, *pues* $\mathcal{I}$ *es modelo de* $p \leftrightarrow t$, *de donde* $\mathcal{I}(s) = 1$ *dado que* $\mathcal{I}$ *también es modelo de* $t \leftrightarrow s$. *Finalmente, como* $\mathcal{I}(s \to r) = 1$ *e* $\mathcal{I}(s) = 1$, *entonces* $\mathcal{I}(r) = 1$. *Por lo tanto,* $\mathcal{I}(q \to r) = 1$.

Ejemplo 7. *Considérese el siguiente conjunto* $\Gamma = \{(p \wedge q), (q \vee r), (\neg s)\}$. *Muestre que* $\Gamma \models p \wedge s$.
Solución: Debemos mostrar que $\mathcal{I}(p \wedge s) = 1$. *Entonces*

$\mathcal{I}(p \wedge q) = 1$	*Premisa (1)*
$\mathcal{I}(q \vee r) = 1$	*Premisa (2)*
$\mathcal{I}(\neg s) = 1$	*Premisa (3)*
$\mathcal{I}(p) = 1$	*por (1)*
$\mathcal{I}(q) = 1$	*por (1)*
$\mathcal{I}(s) = 0$	*por (3)*

De manera que la interpretación dada por $\mathcal{I}(p) = \mathcal{I}(q) = 1$ *e* $\mathcal{I}(s) = 0$ *e* $\mathcal{I}(r) =$ *arbitrario, es un contraejemplo al argumento, pues con esta interpretación tenemos que* $\mathcal{I}(p \wedge s) = 0$. *Por lo tanto, el argumento es falso.*

Proposición 2. *La relación de consecuencia lógica cumple las siguientes propiedades:*

- *Si* $\varphi \in \Gamma$ *entonces* $\Gamma \models \varphi$.
- *Principio de refutación:* $\Gamma \models \varphi$ *si y sólo si* $\Gamma \cup \{\neg\varphi\}$ *es insatisfacible.*
- $\Gamma \models \varphi \to \psi$ *si y sólo si* $\Gamma \cup \{\varphi\} \models \psi$.
- *Insatisfacibilidad implica trivialidad: Si* Γ *es insatisfacible entonces* $\Gamma \models \varphi$ *para toda* $\varphi \in PROP$.
- *Si* $\Gamma \models \bot$ *entonces* Γ *es insatisfacible.*
- $\varphi \equiv \psi$ *si y sólo si* $\varphi \models \psi$ *y* $\psi \models \varphi$.
- $\models \varphi$ *(es decir, si* φ *es tautología) si y sólo si* $\varnothing \models \varphi$ *(es decir,* φ *es consecuencia lógica del conjunto vacío).*

3. Deducción Natural

Los sistemas de deducción natural, introducidos por G. Gentzen en 1934, son formalismos deductivos que modelan el razonamiento matemático ordinario de manera más fiel que un sistema axiomático o que el método de tableaux.
Un sistema de deducción natural consiste de reglas de inferencia donde las hipótesis se encuentran en la parte superior de una línea horizontal y la conclusión en la parte inferior. Las reglas describen las formas para **introducir** y **eliminar** cada uno de los conectivos lógicos. Las pruebas o derivaciones que se construyen en estos sistemas son mediante la aplicación de dichas reglas en una sucesión adecuada que relaciona conclusiones con premisas de reglas posteriores. De igual forma que en el razonamiento ordinario se pueden hacer hipótesis temporales durante la prueba, las cuales se pueden **descargar** al incorporarlas a la conclusión.

Definición 6. *Un contexto es un conjunto finito de fórmulas* $\{\varphi_1, ..., \varphi_n\}$. *Usualmente denotaremos un contexto con* Γ, Δ. *En lugar de* $\Gamma \cup \Delta$ *escribimos* Γ, Δ. *Análogamente,* Γ, φ *denota al contexto* $\Gamma \cup \{\varphi\}$.

Así, siempre que un contexto sea de la forma Γ, A, suponemos que la fórmula A no figura en Γ.

3.1. Reglas de inferencia

La relación de derivabilidad o deducibilidad $\Gamma \vdash A$ se define recursivamente a partir de la regla de inicio

$$\frac{}{\Gamma, \varphi \vdash \varphi}\ (\text{Hip})$$

dando reglas de introducción y eliminación para cada conectivo que queremos esté presente en el sistema:

- Implicación:

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\ (\to \text{I}) \qquad \frac{\Gamma \vdash \varphi \to \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\ (\to \text{E})$$

- Conjunción:

$$\frac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\,(\wedge \text{ I}) \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}\,(\wedge \text{ E}) \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\,(\wedge \text{ E})$$

- Disyunción:

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\,(\vee \text{ I}) \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}\,(\vee \text{ I}) \qquad \frac{\Gamma \vdash \varphi \vee \psi \qquad \Gamma, \varphi \vdash \chi \qquad \Gamma, \psi \vdash \chi}{\Gamma \vdash \chi}\,(\vee \text{ E})$$

Notemos que mediante estas reglas de inferencia no estamos derivando fórmulas sino expresiones de la forma $\Gamma \vdash A$, conocidas como secuentes. En particular las reglas de inferencia son correctas con respecto a la consecuencia lógica, es decir, transforman secuentes válidos (respecto a $\models$) en secuentes válidos como lo asegura lo siguiente

Proposición 3. *Sean Γ un contexto y A, B, C fórmulas. Se cumple lo siguiente*

- *Si $\Gamma, A \models B$ entonces $\Gamma \models A \to B$.*
- *Si $\Gamma \models A$ y $\Gamma \models A \to B$ entonces $\Gamma \models B$.*
- *$\Gamma \models A \wedge B$ si y sólo si $\Gamma \models A$ y $\Gamma \models B$.*
- *Si $\Gamma \models A$ entonces $\Gamma \models A \vee B$.*
- *Si $\Gamma \models B$ entonces $\Gamma \models A \vee B$.*
- *Si $\Gamma \models A \vee B$, $\Gamma, A \models C$ y $\Gamma, B \models C$ entonces $\Gamma \models C$.*

Definición 7. *Una derivación del secuente $\Gamma \vdash A$ es una sucesión finita de secuentes de la forma $\Gamma_1 \vdash A_1, ..., \Gamma_n \vdash A_n$ tal que:*

- *$\Gamma_i \vdash A_1$ es una instancia de la regla (Hip) ó*
- *$\Gamma_i \vdash A_1$ es conclusión de alguna regla de inferencia tal que las premisas necesarias figuran antes en la sucesión.*
- *$\Gamma \vdash A$ es el último elemento de la sucesión.*

Definición 8. *Si $\vdash \varphi$ es derivable, es decir, si $\varnothing \vdash \varphi$ es derivable (φ es derivable sin hipótesis) entonces decimos que φ es un teorema.*

3.2. La Negación

Un sistema de deducción natural puede clasificarse, de acuerdo a qué clase de negación tenga, como minimal, intuicionista o clásico.

3.2.1. Lógica Minimal

Se dice que la lógica es minimal si no hay reglas para la negación $\neg$ ni para lo falso $\bot$. En un sistema minimal, la constante $\bot$ está presente pero no tiene propiedades particulares. En la presencia de $\bot$, el símbolo de negación se define como

$$\neg A =_{def} A \to \bot$$

En cuyo caso hablamos de la negación constructiva, cuyas reglas de inferencia son:

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\,(\neg\ \mathrm{I}) \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash \bot}\,(\neg\ \mathrm{E})$$

3.2.2. Lógica Intuicionista

La lógica intuicionista se obtiene al agregar a la lógica minimal la regla de eliminación de lo falso ($\bot$ E) conocida también como ex-falso-quodlibet.

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}\,(\mathrm{EFQ})$$

Se observa que cualquier fórmula derivada en la lógica minimal sigue siendo derivable en la lógica intuicionista.

3.2.3. Lógica Clásica

Para recuperar a la lógica clásica tenemos que postular alguna de las siguientes reglas:

- Tercer excluido.

$$\frac{}{\Gamma \vdash A \vee \neg A}\,(\mathrm{TE})$$

- Reducción al absurdo.

$$\frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A}\,(\mathrm{RAA})$$

- Eliminación de la doble negación.

$$\frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}\,(\neg\neg\ \mathrm{E})$$

3.3. Ejemplos de derivaciones

Ejemplo 8. *Demuestra que el siguiente secuente es válido usando deducción natural.*

$$\{p \to (q \vee r)\} \vdash (p \to q) \vee (p \to r)$$

Demostración. Por el principio de refutación basta mostrar que

$$\Gamma = \{p \to (q \vee r), (p \wedge \neg q) \wedge (p \wedge \neg r)\} \vdash \bot.$$

Entonces

$1.\Gamma \vdash p \to (q \vee r)$	Hip
$2.\Gamma \vdash (p \wedge \neg q) \wedge (p \wedge \neg r)$	Hip
$3.\Gamma \vdash p \wedge \neg q$	($\wedge$ E) 2
$4.\Gamma \vdash p \wedge \neg r$	($\wedge$ E) 2
$5.\Gamma \vdash p$	($\wedge$ E) 3
$6.\Gamma \vdash \neg q$	($\wedge$ E) 3
$7.\Gamma \vdash \neg r$	($\wedge$ E) 4
$8.\Gamma \vdash q \vee r$	($\to$ E) 1 y 5
$9.\Gamma, q \vdash q$	Hip
$10.\Gamma, q \vdash \bot$	($\neg$ E) 6 y 9
$11.\Gamma, r \vdash r$	Hip
$12.\Gamma, r \vdash \bot$	($\neg$ E) 7 y 11

Así, como en ambos casos obtenemos $\bot$ podemos concluir que $\{p \to (q \vee r)\} \vdash (p \to q) \vee (p \to r)$.

□

Ejemplo 9. *Demuestra que el siguiente secuente es válido usando deducción natural.*

$$\vdash (p \wedge q \to r) \to p \to q \to r$$

Demostración. Aplicando varias veces el Teorema de Deducción natural basta mostrar que

$$\Gamma = \{(p \wedge q) \to r, p, q\} \vdash r$$

Entonces

$1.\Gamma \vdash (p \wedge q) \to r$	Hip
$2.\Gamma \vdash p$	Hip
$3.\Gamma \vdash q$	Hip
$4.\Gamma \vdash p \wedge q$	($\wedge$ I) 2 y 3
$5.\Gamma \vdash r$	($\to$ E) 1 y 4

Por lo tanto, podemos concluir que $\vdash (p \wedge q \to r) \to p \to q \to r$.

□

Ejemplo 10. *Demuestra que el siguiente secuente es válido usando deducción natural.*

$$\Gamma = \{p, p \to q, p \to (q \to r)\} \vdash r$$

Demostración.

1.$\Gamma \vdash p$	Hip
2.$\Gamma \vdash p \to q$	Hip
3.$\Gamma \vdash p \to (q \to r)$	Hip
4.$\Gamma \vdash q$	($\to$ E) 1 y 2
5.$\Gamma \vdash q \to r$	($\to$ E) 1 y 3
6.$\Gamma \vdash r$	($\to$ E) 4 y 5

Por lo tanto, podemos concluir que $\{p, p \to q, p \to (q \to r)\} \vdash r$

□

3.4. Teorema de completud y correctud para la lógica clásica

El siguiente teorema vincula el mundo de la semántica con el de la sintáxis de manera biunívoca. Estrictamente hablando, se conoce como **completud** a la implicación directa, mientras que al regreso se le conoce como **correcctud**, ya que asegura que no se pueden deducir cosas falsas.

Teorema 1. *Sean Γ un conjunto de fórmulas y φ una fórmula.*

$$\Gamma \models \varphi \textit{ si y sólo si } \Gamma \vdash \varphi$$

Demostración. Sean Γ un conjunto de fórmulas y φ una fórmula.

$\Rightarrow$) Supongamos que $\Gamma \models \varphi$. Luego, para toda interpretación $\mathcal{I}$ tal que $\mathcal{I}(\psi) = 1$ para toda $\psi \in \Gamma$, se da $\mathcal{I}(\varphi) = 1$. Esto equivale a decir que no hay interpretación tal que $\mathcal{I}(\psi) = 1$ para toda $\psi \in \Gamma$ y $\mathcal{I}(\varphi) = 0$, lo cual implica que no se puede dar $\Gamma \not\vdash \varphi$, es decir, obtenemos que $\Gamma \vdash \varphi$.

$\Leftarrow$) Inducción sobre $\Gamma \vdash \varphi$. Lo cual equivale a probar todas las reglas del sistema de lógica clásica preservan la noción $\models$, pero esto es justo lo que dice la proposición 3.

□

4. El problema a resolver.

Se ha cometido un asesinato (sólo hay un asesino). Se sospecha del esposo, del amante y del mayordomo. Durante los interrogatorios cada sospechoso hizo 2 declaraciones clave:

- Esposo.
  1. Yo no lo hice.
  2. El mayordomo tampoco lo hizo.

- Mayordomo.
    1. El esposo no lo hizo.
    2. Lo hizo el amante.
- Amante.
    1. Yo no lo hice.
    2. Lo hizo el esposo.

Al final del juicio pudimos enterarnos de que uno de los sospechosos era un lógico que había dicho la verdad en sus dos declaraciones, otro sospechoso resultó ser un estafador ya que mintió en ambas declaraciones. El tercer sospechoso resultó ser un loco que dijo la verdad en una declaración, pero mintió en otra. El objetivo es determinar quién es el asesino, quién es el lógico, quién es el estafador y quién es el loco.

5. Solución del problema lógico.

5.1. Implementación de la solución.

Se crearon dos programas para este proyecto. Explicaremos detalladamente el propósito de cada programa y sus funciones, aunque esto también se encuentra en el programa, incluyendo un ejemplo de entrada y salida de cada función.

1) **LogicaProp.hs**
Importamos la biblioteca `Data.List` para poder utilizar la función `union` más adelante. Una variable proposiconal será del tipo `Char` y un estado será una lista de tuplas donde el primer componente de la tupla es una variable proposicional y su segundo componente será el valor booleano asociado a dicha variable. Para su implementación funcional, creamos los sinónimos de tipo para `VarP` como sinónimo de `Char`, y `Estado` como sinónimo de `[(VarP, Bool)]`. Creamos el tipo de dato para las fórmulas proposicionales, el cual definimos de la siguiente forma:

```
data Prop = Var VarP -- Var 'p'
          | Neg Prop  -- ~P
          | Conj Prop Prop -- P ^ Q
          | Disy Prop Prop -- P v Q
          | Impl Prop Prop -- P -> Q
          | Syss Prop Prop -- P <-> Q
          deriving (Eq, Ord, Show)
```

donde `Var VarP = Var Char`, y los demás son los conectivos lógicos (binarios y unarios) que ya definimos anteriormente.
También se dan ejemplos de variables proposicionales y fórmulas como referencias al lector de cómo se deben escribir y pueda utilizar el programa con mayor claridad en un futuro.
En seguida se encuentran las funciones de lógica proposicional, las cuales son:

- **Función interp**. Recibe una fórmula φ y un estado e. Regresa la interpretación de φ con el estado dado.
  La función está implementada de la siguiente forma:

```
interp :: Prop -> Estado -> Bool
interp phi e = case phi of
    Var i -> buscaBool i e
    Neg p -> not (interp p e)
    Conj p q -> (interp p e) && (interp q e)
    Disy p q -> (interp p e) || (interp q e)
    Impl p q -> (not (interp p e)) || (interp q e)
    Syss p q -> (interp p e) == (interp q e)
```

  la cual es una aplicación directa de nuestra definición de semántica. Aquí podemos notar dos cosas: la primera, que se utilizó una equivalencia lógica para obtener la interpretación de la implicación; y la segunda, que utilizamos una función auxiliar `buscaBool`. Ésta recibe una variable proposicional `varP`, y un estado `[(varP,Bool)]`. Regresa la segunda componente del primer par ordenado de la lista de estados l, cuyo primer componente sea igual a la variable `varP`. Es decir, regresa el valor booleano asociado a la primer aparición de la variable proposicional `varP` que se le pase como parámetro.
  La función auxiliar está implementada de la siguiente forma:

```
buscaBool :: (Eq varP) => varP -> [(varP, Bool)] -> Bool
buscaBool varP e = head [b | (x,b) <- e, varP == x]
```

  Esta función en particular se encuentra hasta abajo del código, en el apartado de *Funciones auxiliares.*

- **Función vars**. Recibe una formula φ. Regresa la lista de variables proposicionales que figuran en φ, sin repetición.
  La función está implementada de la siguiente forma:

```
vars :: Prop -> [VarP]
vars phi = case phi of
    Var x -> [x]
    Neg p -> vars p
    Conj p q -> vars p `union` vars q
    Disy p q -> vars p `union` vars q
    Impl p q -> vars p `union` vars q
    Syss p q -> vars p `union` vars q
```

  Aquí simplemente utilizamos la función `union` para unir las listas finales que contienen las variables proposicionales de cada una de las fórmulas. Por cómo está definida `union`, nos regresa la lista que contiene todos los elementos de las listas que recibe, sin repetición.

- **Función estados**. Recibe una fórmula φ con $n-$variables proposicionales. Regresa la lista con los 2^n estados distintos para φ.
  La función está implementada de la siguiente forma:

```
estados:: Prop -> [Estado]
estados phi = subconj (vars phi)
    where subconj [] = [[]]
          subconj (x:xs) =
            [(x,True):i | i <- subconj xs] ++ [(x,False):i | i <- subconj xs]
```

  Notemos que para obtener los 2^n estados posibles, debemos obtener el subconjunto de listas de la lista de variables proposicionales de φ, hacer las combinaciones posibles entre los estados y los valores booleanos e ir concatenando las parejas de tuplas para formar

una lista (que es justo lo que hacemos en la función subconj). Y como subconj trabaja con una lista de comprensión, al final obtendremos una lista de listas (ya que los estados son listas) con los 2^n estados.

- **Función varCN**. Recibe una fórmula φ. Regresa la lista de variables proposicionales que figuran en φ junto con su conectivo unario (si es que lo tiene).
  La función está implementada de la siguiente forma:

```
varCN :: Prop -> [Prop]
varCN phi = case phi of
Var x -> [Var x]
(Neg (Var i)) -> [Neg (Var i)]
Neg p -> varCN p
Conj p q -> varCN p `union` varCN q
Disy p q -> varCN p `union` varCN q
Impl p q -> varCN p `union` varCN q
Syss p q -> varCN p `union` varCN q
```

  La diferencia con la función vars es que si la variable porposicional tiene una negación, la manda a la lista junto con su conectivo unario.
  Aquí simplemente utilizamos la función `union` para unir las listas finales que contienen las variables proposicionales de cada una de las fórmulas. Por cómo está definida `union`, nos regresa la lista que contiene todos los elementos de las listas que recibe, sin repetición.

- **Función sonDiferentes**. Recibe una fórmula φ con únicamente dos variables proposicionales y un estado e. Nos dice si la interpretación de cada una de sus variables con el estado e son diferentes.
  La función está implementada de la siguiente forma:

```
sonDiferentes :: Prop -> Estado -> Bool
sonDiferentes phi e = (interp (head (varCN phi)) e) /= (interp (last (varCN phi)) e)
```

  En esta función es donde `varCN` cobra mucha importancia. Sabemos que las variables proposicionales de φ pueden estar solas o negadas, por lo que no nos sirve usar la función `vars` ya que no estaríamos considerando el caso en que las variables están negadas. Así, utilizamos la función `varCN` para evaluar ambas variables de la fórmula φ y verificar si la interpretación de éstas es diferente o no. Esta función es fácil de realizar ya que únicamente hay que checar si la interpretación de la primer variable de φ es diferente a la interpretación de la segunda variable de φ, con el estado dado.

- **Función sonIguales** Recibe una fórmula φ con únicamente dos variables proposicionales y un estado e. Nos dice si la interpretación de cada una de sus variables con el estado e son iguales.
  La función está implementada de la siguiente forma:

```
sonIguales :: Prop -> Estado -> Bool
sonIguales beta e = (interp (head (varCN beta)) e) == (interp (last (varCN beta)) e)
```

  En esta función es donde `varCN` cobra mucha importancia. Sabemos que las variables proposicionales de φ pueden estar solas o negadas, por lo que no nos sirve usar la función `vars` ya que no estaríamos considerando el caso en que las variables están negadas. Así, utilizamos la función `varCN` para evaluar ambas variables de la fórmula φ y verificar si la interpretación de éstas es igual o no. Esta función es fácil de realizar ya que únicamente hay que checar si la interpretación de la primer variable de φ es igual a la interpretación de la segunda variable de φ, con el estado dado.

2) **Proyecto1.hs**
Importamos el módulo del programa anterior con `import LogicaProp`, y nuevamente importamos la biblioteca `Data.List` para poder utilizar la función `intersect` más adelante.
Definimos las variables proposicionales que vamos a utilizar para resolver el problema de la siguiente manera

- `p` = Lo hizo el esposo.
- `q` = Lo hizo el amante.
- `r` = Lo hizo el mayordomo.

las cuales corresponden a las proposiciones atómicas de las declaraciones de los sospechosos. El conjunto de declaraciones de los sospechosos queda como:

- Declaración del esposo: $\{\neg p, \neg r\}$
- Declaración del mayordomo: $\{\neg p, q\}$
- Declaración del amante: $\{\neg q, p\}$

Y por la proposición 1 del reporte sabemos que podemos reescrbir estos conjuntos de declaraciones como:

- `desposo = (Conj (Neg (Var p)) (Neg (Var r)))` $\equiv \neg p \wedge \neg r$
- `damante = (Conj (Neg (Var p)) (Var q))` $\equiv \neg p \wedge q$
- `demayordomo = (Conj (Neg (Var q)) (Var p))` $\equiv \neg q \wedge p$

Finalmente, definimos la conjunción de la declaración de los tres sospechosos

```
argumento = (Conj desposo (Conj dmayordomo damante))
```

Enseguida están las funciones que son propias del proyecto.

- **Función unAsesino** Recibe una fórmula. Regresa los únicos tres estados donde se cumple que sólo hay un asesino.
  La función está implementada de la siguiente forma:

```
unAsesino :: Prop -> [Estado]
unAsesino _ =
    [e | e <- estados argumento,
    (buscaBool p e == True && buscaBool q e == False && buscaBool r e == False)
    || (buscaBool p e == False && buscaBool q e == True && buscaBool r e == False)
    || (buscaBool p e == False && buscaBool q e == False && buscaBool r e == True)]
```

  El problema nos dice que sólo hay un asesino, y de acuerdo a cómo definimos nuestras variables proposicionales p, q y r, entonces en los estados donde se cumple que sólo hay un asesino pasa que p es verdadero y los demás son falsos, ó q es verdadero y los demás falsos, ó r es verdadero y los demás son falsos. Entonces esta función es muy importante pues así sólo estaremos trabajando en los estados que cumplen esta primera condición del problema.
  Para lograr esto, utilizamos una lista de comprensión para obtener la lista con los tres estados que estamos buscando. Los estados los obtenemos de los $2^3 = 8$ estados posibles

de nuestro argumento. Las propiedades en la lista de comprensión usan la función auxiliar `buscaBool` para definir cómo son los valores booleanos de cada una de las variables proposicionales de los estados que estamos buscando. Así garantizamos que buscamos los estados correctos.
Notemos que no nos importa qué fórmula recibamos como entrada (debe ser una fórmula válida para que haskell no llore, así que utilizamos argumento por simplicidad), la función siempre nos arrojará el mismo resultado (que es lo que queremos).

`Salida de la función:`

```
*Main> unAsesino argumento
[[('p',True),('r',False),('q',False)],
[('p',False),('r',True),('q',False)],
[('p',False),('r',False),('q',True)]]
```

- **Función modelos** Recibe una fórmula φ. Regresa la lista con todos los modelos que satisfacen a la fórmula φ.
  La función está implementada de la siguiente forma:

```
modelos :: Prop -> [Estado]
modelos phi = [e | e <- unAsesino argumento, interp phi e == True]
```

  Para obtener los modelos utilizamos una lista de comprensión, la cual tiene como propiedad que la interpretación de φ con los estados de `unAsesino` sean igual a `True` (y como vimos en la parte teórica, esto nos dice que si se cumple la propiedad descrita anteriormente, entonces ese es un modelo de φ). Notemos que en lugar de tomar los estados de la fórmula φ, tomamos los estados obtenidos de la función `unAsesino` con el argumento. ¿Por qué? Bueno, esta función unicamente la usaremos para obtener los modelos de cada una de las declaraciones de los sospechosos, y como los únicos estados que nos interesan son los de `unAsesino`, entonces utilizamos éstos.
  Esta función será de mucha utilidad en la función `unaVerdad` ya que ahí empezamos a descartar estados que no cumplan la propiedad de que sólo una persona dice la verdad. Los estados que queden implican que existe al menos el personaje lógico.

  `Salida de la función:`

```
*Main> modelos desposo
[[('p',False),('r',False),('q',True)]]

*Main> modelos dmayordomo
[[('p',False),('r',False),('q',True)]]

*Main> modelos damante
[[('p',True),('r',False),('q',False)]]
```

- **Función noModelos**. Recibe una fórmula φ. Regresa la lista con todos los estados que no satisfacen a la fórmula φ.
  La función está implementada de la siguiente forma:

```
noModelos :: Prop -> [Estado]
noModelos phi =
    [e | e <- unAsesino argumento, interp phi e == False]
```

  Para obtener los no modelos utilizamos una lista de comprensión, la cual tiene como propiedad que la interpretación de φ con los estados de `unAsesino` sean igual a `False`

(y como vimos en la parte teórica, esto nos dice que si se cumple la propiedad descrita anteriormente, entonces ese no es un modelo de φ). Notemos que en lugar de tomar los estados de la fórmula φ, tomamos los estados obtenidos de la función `unAsesino` con el argumento. ¿Por qué? Bueno, esta función unicamente la usaremos para obtener los no modelos de cada una de las declaraciones de los sospechosos, y como los únicos estados que nos interesan son los de `unAsesino`, entonces utilizamos éstos.
Esta función será de mucha utilidad en la función `unaVerdad` ya que ahí empezamos a descartar estados que no cumplan la propiedad de que sólo una persona dice la verdad. Los estados que queden implican que existe al menos el personaje lógico.

`Salida de la función:`

```
*Main> noModelos desposo
[[('p',True),('r',False),('q',False)],
[('p',False),('r',True),('q',False)]]

*Main> noModelos dmayordomo
[[('p',True),('r',False),('q',False)],
[('p',False),('r',True),('q',False)]]

*Main> noModelos damante
[[('p',False),('r',True),('q',False)],
[('p',False),('r',False),('q',True)]]
```

- **Función unaVerdad**. Recibe una fórmula. Regresa la lista de estados que satisfacen a una de las declaraciones de los sospechosos y al resto no.
La función está implementada de la siguiente forma:

```
unaVerdad :: Prop -> [Estado]
unaVerdad _ =
    [i | i <- interseccion (modelos desposo)
    (noModelos dmayordomo) (noModelos damante)]
    ++ [i | i <- interseccion (modelos dmayordomo)
    (noModelos desposo) (noModelos damante)]
    ++ [i | i <- interseccion (modelos damante)
    (noModelos desposo) (noModelos dmayordomo)]
```

El problema nos dice que hay una persona que dice dos verdades, por lo que es buena idea comenzar descartando todos aquellos estados que no cumplen la propiedad de tener sólo una persona que dice la verdad (ya que de lo contrario no se cumple que exista un lógico, un estadafor y un loco). Esto lo logramos mediante la intersección de los modelos de una declaracion y los noModelos de las otras dos declaraciones. Así, nuestra lista de comprensión tiene como propiedad todos los posibles casos de la condición descrita anteriormente.
Por ejemplo, sabemos que el estado `[('p',False),('r',False),('q',True)]` satisface a la declaracion del esposo. Gracias a esto ya tenemos al personaje lógico (ya que si lo satisface, entonces dice dos verdades pues el conectivo de las declaraciones es la conjunción). Entonces, este estado no tendría que ser un modelo para la declaración del mayordomo y del amante. Si este estado cumple lo anterior, entonces cumple la propiedad de que al menos existe un lógico y ya sólo tendríamos que comprobar la existencia del loco y del estafador.
Notemos que no nos importa qué fórmula recibamos como entrada (debe ser una fórmula válida para que haskell no llore, así que utilizamos argumento por simplicidad), la función siempre nos arrojará el mismo resultado (que es lo que queremos).

Salida de la función:

```
*Main> unaVerdad argumento
[[('p',True),('r',False),('q',False)]]
```

Este resultado se puede corroborar con los resultados dados en las funciones anteriores de modelos y noModelos.
Ahora, veamos el auxiliar que utilizamos aqui. La función `interseccion` recibe tres listas `xs, ys, zs`. Regresa una lista con los elementos que tienen en común las tres listas.
La implementación de la función es de la siguiente forma:

```
interseccion :: Eq a => [a] -> [a] -> [a] -> [a]
interseccion [] [] [] = []
interseccion xs ys zs = intersect (intersect xs ys) zs
```

Para nuestro caso particular, la utilizamos para obtener la intersección de tres listas de listas en la función `unaVerdad`. Notemos que aquí utilizamos la función intersect, ya que nos ahorra tener que hacer a pie la función de intersección. Nuestra función simplemente es un caso particular que necesitamos para el proyecto.
Esta función la podemos encontrar hasta abajo del código, en el apartado de *Funciones auxiliares*.

Ahora que tenemos el estado que cumple que sólo hay un asesino y sólo una persona dice la verdad, entonces sólo nos interesa saber si este estado cumple con la propiedad de tener un loco y un estafador. Para esto tenemos la siguiente función:

- **Función juicio**. Recibe una fórmula. Regresa la lista con los estados que cumplen la propiedad de que exista un lógico (dos verdades), un estafador (dos mentiras) y un loco (una verdad y una mentira).
La implementación de la función es de la siguiente manera:

```
juicio:: Prop -> [Estado]
juicio _ =
    [e | e <- unaVerdad argumento,
    (sonDiferentes desposo e == True && sonIguales dmayordomo e == True)
    && (sonIguales dmayordomo e == True && sonIguales damante e == True)
    || (sonIguales desposo e == True && sonDiferentes dmayordomo e == True)
    && (sonDiferentes dmayordomo e == True && sonIguales damante e == True)
    || (sonIguales desposo e == True && sonIguales dmayordomo e == True)
    && (sonIguales dmayordomo e == True && sonDiferentes damante e == True)]
```

Expliquemos cómo funciona juicio. Tomamos los estado que obtuvimos de la función `unaVerdad`, el cuál sólo es uno (pero aún así hay que verificar que ese estado cumple con las propiedades del problema). Y como sabemos que en este estado sólo una persona dice la verdad, entonces debemos buscar al estafador y al loco, los cuales tienen una característica muy peculiar: la interpretación de cada una de las variables proposicionales de la declaración de éstos dos personajes con el estado de la función 'unaVerdad' debería ser diferente (si es el loco) e igual (si es el estafador). Entonces, recordemos que sabemos que hay un lógico, el cual dice dos verdades y por lo tanto la interpretación de cada una de sus variables es igual (por ser conjunción) con el estado de `unaVerdad`. Sabiendo esto, basta considerar tres casos simples en nuestra lista de comprensión para deducir que existen los tres personajes solicitados. Veremos sólo el primer caso, ya que los demás son análogos.
Supongamos que la interpretación de las variables de la declaracion del esposo con el estado

de 'unaVerdad' es diferente y la interpretación de las variables de la declaración del mayordomo con el estado de `unaVerdad` es igual. Entonces sabemos que el esposo es el loco (ya que dice una verdad y una mentira) y el mayordomo puede ser el estafador o el lógico. Pero si además, la interpretación de las variables proposicionales de la declaración del amante con el estado de `unaVerdad` son iguales entonces significa que el amante también puede ser el estafador ó el lógico. Y como ese estado de `unaVerdad`, es el mismo que aplicamos en las tres declaraciones, entonces sabemos que ese estado es el que cumple la propiedad que estamos buscando. Bastaría revisar manualmente quién es nuestro lógico y nuestro estafador.

Notemos que no nos importa qué fórmula recibamos como entrada (debe ser una fórmula válida para que haskell no llore, así que utilizamos argumento por simplicidad), la función siempre nos arrojará el mismo resultado (que es lo que queremos).

Salida de la función:

```
*Main> juicio argumento
[[('p',True),('r',False),('q',False)]]
```

5.2. ¿Quién es el asesino?

Abrimos nuestra terminal y nos movemos al directorio donde se encuentren los archivos **LogicaProp.hs** y **Proyecto1.hs**. Una vez que estemos en el directorio deseado, ejecutamos el comando `ghci`, con el cual obtenemos el siguiente resultado:

```
GHCi, version 8.6.3: http://www.haskell.org/ghc/ :? for help
Prelude>
```

Luego, compilamos el programa principal **Proyecto1.hs** con el comando `:l Proyecto1`

```
Prelude> :l Proyecto1
[1 of 2] Compiling LogicaProp   ( LogicaProp.hs, interpreted )
[2 of 2] Compiling Main         ( Proyecto1.hs, interpreted )
Ok, two modules loaded.
```

Para conocer quién es el asesino, ejecutamos la función **juicio** del programa *Proyecto1.hs* con el parámetro **argumento**

```
*Main> juicio argumento
```

donde *argumento* es la conjunción de las declaraciones de los tres sospechosos. Dicha ejecución nos arroja el siguiente resultado

```
[[('p',True),('r',False),('q',False)]]
```

Este es el estado donde sólo hay un asesino, un lógico, un estafador y un loco. Con el simple estado sabemos que `p = True`, y recordando nuestra implementación, sabemos que p corresponde a la proposición *Lo hizo el esposo*. Por lo tanto, **el esposo es el asesino**.
Finalmente, para saber quién es el loco, quién es el estafador y quién es el lógico, basta realizar manualmente la interpretación de cada una de las declaraciones de los sospechosos con el estado que obtuvimos anteriormente. Así,

- Declaración del esposo: $\{\neg p, \neg r\}$.
  Como $\mathcal{I}(p) = 1$ y $\mathcal{I}(r) = 0$, entonces $\mathcal{I}(\neg p) = 0$ y $\mathcal{I}(\neg r) = 1$. Por lo tanto, el esposo es el **loco**.

- Declaración del mayordomo: $\{\neg p, q\}$.
  Como $\mathcal{I}(p) = 1$ y $\mathcal{I}(q) = 0$, entonces $\mathcal{I}(\neg p) = 0$. Por lo tanto, el mayordomo es el **estafador**.

- Declaración del amante: $\{\neg q, p\}$.
  Como $\mathcal{I}(q) = 0$ y $\mathcal{I}(p) = 1$, entonces $\mathcal{I}(\neg q) = 1$. Por lo tanto, el amante es el **lógico**.

Con esto, hemos resuelto por completo el problema.