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

Rubí Rojas Tania Michelle

24 de marzo de 2019

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. Ejemplos de proposiciones:

- Los números pares son divisibles por dos.
- La música clásica es la más antigua del mundo.
- Una ballena no es roja.
- He pasado mis vacaciones en Grecia.

Ejemplos de enunciados que no son proposiciones:

- ¡Auxilio, me desmayo!
- ¿Qué día es hoy?
- 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 $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \to \psi), (\varphi \leftrightarrow \psi) \in PROP$.
- Son todas.

Ahora veamos un par de ejemplos:

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

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

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 $\mathcal{I}^\star$.
Ahora veamos un par de ejemplos:

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

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

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 $\varphi \in \Gamma$.

Ahora, veamos ejemplos de lo anterior:

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) $\Gamma = \{p \rightarrow q, r \rightarrow s, \neg s\}$.
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) $\varphi = p \rightarrow (q \vee r)$
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) $\Gamma = \{p \rightarrow q, \neg(q \vee s), s \vee p\}$
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 $\mathcal{I}(p \rightarrow q) = 1$, 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.

1.4. Especificación formal de propiedades y teoremas

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 (no tiene operadores).

- $\phi = \top$. Entonces $atom(\top) = 1 = 0 + 1 = con(\top) + 1$
- $\phi = \bot$. Entonces $atom(\bot) = 1 = 0 + 1 = con(\bot) + 1$
- $\phi = VarP$. Entonces $atom(VarP) = 1 = 0 + 1 = con(VarP) + 1$

Hipótesis de inducción: Supongamos que

$$atom(\phi') \leq con(\phi') + 1 \text{ y } atom(\phi'') \leq con(\phi'') + 1$$

Paso inductivo: Probamos la propiedad para dos casos:

i) $\phi = \neg\phi'$. Sabemos que $atom(\phi) = atom(\phi')$ y $con(\phi) = con(\phi')$. Entonces,

$$\begin{aligned} atom(\phi) &= atom(\neg\phi') && \text{def. de } \phi \text{ en el caso } i) \\ &= atom(\phi') && \text{def. recursiva de } \phi \\ &\leq con(\phi') + 1 && \text{hipótesis de inducción} \\ &\leq con(\phi) + 1 && \text{ya que } con(\phi) = con(\phi') \end{aligned}$$

ii) $\phi = (\varphi \star \psi)$. Entonces tenemos que

$$\begin{aligned} atom(\phi) &= atom(\varphi \star \psi) && \text{def. de } \phi \text{ en el caso } ii) \\ &= atom(\varphi) + atom(\psi) && \text{def. recursiva de } atom \\ &\leq (con(\varphi) + 1) + (con(\psi) + 1) && \text{hipótesis de inducción} \\ &\leq (con(\varphi) + con(\psi) + 1) + 1 && \text{reagrupando} \\ &\leq con(\varphi \star \psi) + 1 && \text{def. recursiva de } con \\ &\leq con(\phi) + 1 && \text{def. de } \phi \text{ en el caso } ii) \end{aligned}$$

□

2. El acertijo 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.

3. Solución del problema lógico.

3.1. 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.

- **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 = Top | Bot | Var VarP | Neg Prop | Conj Prop Prop
| Disy Prop Prop | Impl Prop Prop | Syss Prop Prop
deriving (Eq, Ord, Show)
```

  donde `Top = True`, `Bot = False`, `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 p, y un estado $[(p, b)]$, y regresa la segunda componente del primer par ordenado de la lista de estados I, cuyo primer componente sea igual a la variable p. La función auxiliar está implementada de la siguiente forma:

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

  En un inicio se utilizó `++` para concatenar las variables, pero hacía que fallara la función `estados` pues al parecer concatenada cadenas de más, y así se consideraban más de 2^n casos. Así que buscando una solución al problema, nos encontramos con que la función `union` era la ideal para tener todos los elementos que necesitabamos y así no irnos a Disneylandia.

- **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, y hacer las combinaciones posibles entre los estados y los valores booleanos (que es justo lo que hacemos en la sub-función `subconj`). Así, estamos regresando una lista con las listas de estados posibles para la fórmula φ.

- **Función varCN**. Recibe una fórmula φ. Regresa la lista de variables proposicionales que figuran en φ. 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.
  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 gran utilidad de esta función será explicada en la función donde es utilizada.

- **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.

3.2. ¿Quién es el asesino?

Una vez que estamos dentro de la capeta **proyecto1**, compilamos el programa.

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

Luego, ejecutamos el programa con el siguiente comando

```
*Main> juicio declaracionFinal
```

donde *declaracionFinal* es la conjunción de la declaración de los tres sospechosos. Dicha ejecución nos arroja el siguiente resultado

```
*Main>
```

Este es el estado donde un sospechoso dice dos verdades, otro dice una verdad y el último dice dos mentiras. Comparando nuestro resultado con nuestra implementación del problema, es fácil ver que las dos primeras tuplas pertenecen a las declaración del esposo, las dos tuplas que le siguen pertenecen a las declaración del mayordomo, y las últimas dos tuplas pertenecen a la declaración del amante. Con esto podemos concluir que

- El amante es el lógico.
- El esposo es el loco.
- El mayordomo es el estafador.

Y como el amante es el lógico, entonces sabemos que sus dos declaraciones son verdaderas. Por lo tanto, **el esposo es el asesino**.

4. Referencias