UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

ESTRUCTURAS DISCRETAS 2020-1

Profesor: M. en C. I. C. Odín Miguel Escorza Soria
Ayudante de laboratorio: Salazar González Edwin Max

Práctica Siete.

LÓGICA PROPOSICIONAL

1. Objetivos de la práctica

Reforzar conceptos de la lógica proposicional.

- Reforzar sintáxis y semántica de la lógica proposicional.

2. Desarrollo de la práctica

Una proposición es una afirmación que puede ser verdadera o falsa pero no ambas. A dichos valores (verdadero o falso) se les conoce como constantes proposicionales; también es posible representarlos como 1 ó 0, respectivamente. En la vida cotidiana existen cosas que no son verdaderas pero tampoco falsas, ese tipo de cosas no entran en la lógica proposicional [2].

Una proposición atómica es aquella proposición que no puede subdividirse. Las constantes proposicionales también son proposiciones atómicas. Al combinarse dicho tipo de proposiciones se forman proposiciones compuestas. Estas combinaciones se logran mediante el uso de conectivos lógicos [3].

2.1. Sintáxis de la lógica proposicional

La sintáxis se define como reglas desritas en el lenguaje para poder formar proposiciones, dichas reglas están agrupadas en cuatro clases [4]

1. Símbolos de variables: representan proposiciones que no pueden dividirse en otras más simples. Dichos símbolos son variables.

2. Símbolos de conectivos lógicos: se utilizan cuando se representan proposiciones compuestas y son símbolos que conetan proposiciones atómicas u otras proposiciones compuestas. Los conectivos son: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$.

3. Símbolos de puntuación: sirven para definir el orden de evaluación de una expresión. Solo se tienen dos símbolos: paréntesis de apertura "(" y paréntesis de cierre ")".

4. Símbolos de constántes lógicas: estos símbolos son *true* y *false* y representan a los valores verdadero y falso, respectivamente.

A continuación se especifica la sintáxis de la lógica proposicional [1, 4]:

			Denotan:
P	::=	Atom	
Atom	::=	Var, Cons	
P	::=	(P)	
P	::=	■ P	
P	::=	P ⊠ P	
⊠	::=	$\wedge$	conjunción
		$\vee$	disyunción
		$\rightarrow$	implicación
		$\leftrightarrow$	equivalencia lógica
■	::=	¬	negación
Var	::=	*a*, *b*, ...	variables
Cons	::=	true, false	constantes proposicionales.

2.2. Semántica de la lógica proposicional

Una vez definida la sintáxis es el turno de la semántica. La semántica es la relación que se da entre símbolos y su significado [6].

El criterio para asignar valores de *verdadero* (V) y $falso$ (F) a los conectivos lógicos se hace mediante sus valores de verdad para que de igual forma las fórmulas bien formadas tengan una asignación de valores [4].

2.2.1. Negación

Sea P una proposición, entonces $\neg P$ se denomina la negación de P y será verdadera si P es falsa. La tabla de verdad para la negación (ver la Tabla 1) es la siguiente [3]:

P	$\neg P$
V	F
F	V

Cuadro 1: Tabla de verdad de la negación [3].

Ejemplo 2.1: Sea P la variable que representa la proposición "Hace calor.$^{\text{en}}$tonces $\neg P$ se traduciría como "No hace calor".

2.2.2. Conjunción

Sean P y Q proposiciones entonces $P \wedge Q$ se denomina la conjunción de P y Q y será verdadera si y sólo si tanto P como Q son verdaderas. La tabla de verdad para la conjunción (ver la Tabla 2) es la siguiente [4]:

P	Q	$P \wedge Q$
V	V	V
V	F	F
F	V	F
F	F	F

Cuadro 2: Tabla de verdad de la conjunción [4].

Ejemplo 2.2: Sea P la variable que representa la proposición "Es viernesz Q la que representa la proposición "Hay luna llena". Entonces $P \wedge Q$ puede traducirse como "Es viernes y hay luna llena".

2.2.3. Disyunción

Sean P y Q proposiciones entonces $P \vee Q$ se denomina la disyunción de P y Q y será falsa si y sólo si tanto P como Q son falsas. La tabla de verdad para la disyunción (ver la Tabla 3) es la siguiente [4]

P	Q	$P \vee Q$
V	V	V
V	F	V
F	V	V
F	F	F

Cuadro 3: Tabla de verdad de la disyunción [4].

Ejemplo 2.3: Sea P la variable que representa la proposición "Me voy al trabajoz Q la variable que representa la proposición "Me voy de vacaciones". Entonces $P \vee Q$ significaría "Me voy al trabajo o me voy de vacaciones".

2.2.4. Implicación o condicional

Sean P y Q proposiciones entonces $P \rightarrow Q$ se denomina la implicación de P y Q y será falsa si y sólo si P es verdadero y Q es falso. Teniendo en cuenta la implicación lógica $P \rightarrow Q$ entonces se cumple lo siguiente [1, 2]:

- Su recíproca o inversa es $Q \rightarrow P$
- Su contrapositiva es $\neg Q \rightarrow \neg P$
- Su contrarecíproca es $\neg P \rightarrow \neg Q$

La tabla de verdad para la implicación (ver la Tabla 4) es la siguiente [1, 2]:

P	Q	$P \rightarrow Q$
V	V	V
V	F	F
F	V	V
F	F	V

Cuadro 4: Tabla de verdad de la implicación [1, 2].

Ejemplo 2.4: Sea P la variable que representa la proposición "Me desvelo" Q la variable que representa la proposición "Me levanto tarde". Entonces $P \rightarrow Q$ se traduce como "Si me desvelo, me levanto tarde".

2.2.5. Doble implicación, bicaondicional o equivalencia

Sean P y Q proposiciones, la proposición compuesta $P \leftrightarrow Q$ se denomina la equivalencia de P y Q y será verdadera si y sólo si P y Q tienen los mismos valores. La tabla de verdad para la equivalencia lógica (ver la Tabla 5) es la siguiente [5]:

P	Q	$P \leftrightarrow Q$
V	V	V
V	F	F
F	V	F
F	F	V

Cuadro 5: Tabla de verdad de la equivalencia [5].

Ejemplo 2.5: Sea P la variable que representa la proposición "Apruebo el curso" Q la variable que representa la proposición "Estudio constantemente". Entonces $P \leftrightarrow Q$ se traduciría como "Apruebo el curso si y sólo sí estudio constantemente".

2.2.6. Tablas de verdad

Una tabla de verdad contiene todos los posibles estados de una fórmula. Al estudiar los resultados de una tabla de verdad podemos categorizarlos de la siguiente manera [3]:

- Tautología: cuando el resultado final de una fórmula da verdadero en todos sus estados.
- Contradicción: cuando el resultado final de una fórmula da falso en todos sus estados.
- Contingencia: cuando una fórmula no es tautología ni contradicción.

2.2.7. Equivalencia lógica

Dos expresiones A y B son lógicamente equivalentes si es que tienen siempre los mismos valores de verdad, por lo tanto se puede sustituir una por la otra sin afectar los valores de verdad. Cuando dos fórmulas A y B son lógicamente equivalentes se denota $A \equiv B$ si y solo si $A \leftrightarrow B$ es una

tautología [1–3, 6].

A continuación se enlistan algunas de las leyes de equivalencia más importantes dentro de la lógica proposicional [1–4] (ver la Tabla 6):

$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$ $(A \vee B) \vee C \equiv A \vee (B \vee C)$	Asociatividad
$A \wedge B \equiv B \wedge A$ $A \vee B \equiv B \vee A$	Conmutatividad
$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$ $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$	Distributividad
$A \wedge true \equiv A$ $A \vee false \equiv A$	Identidad
$A \wedge false \equiv false$ $A \vee true \equiv true$	Elemento nulo
$A \wedge A \equiv A$ $A \vee A \equiv A$	Idempotencia
$\neg(A \wedge B) \equiv \neg A \vee \neg B$ $\neg(A \vee B) \equiv \neg A \wedge \neg B$	De Morgan
$P \wedge \neg P \equiv false$	Contradicción
$A \vee \neg A \equiv true$	Tercero excluido
$\neg\neg A \equiv A$	Doble negación
$A \rightarrow B \equiv \neg A \vee B$ $A \leftrightarrow B \equiv (\neg A \vee B) \wedge (\neg B \vee A)$ $A \leftrightarrow B \equiv (A \wedge B) \vee (\neg A \wedge \neg B)$	Equivalencia de operadores
$A \wedge (A \vee B) \equiv A$ $A \vee (A \wedge B) \equiv A$	Ley de absorción
$(A \wedge B) \vee (\neg A \wedge B) \equiv B$ $(A \vee B) \wedge (\neg A \vee B) \equiv B$	Ley de simplificación

Cuadro 6: Tabla de equivalencias lógicas.

2.3. Interpretaciones

Las interpretaciones son un concepto importante dentro de la semántica de la lógica proposicional.

Una interpretación para A es la función I que asigna valores de verdad a los átomos que aparecen en A. Dentro del concepto de interpretaciones los valores 1 y 0 denotarán verdadero y falso respectivamente. Con esto, la función I queda definida como I:Var$\rightarrow\{0,1\}$.

La interpretación de las fórmulas está definida por cada estado I de las variables de dicha fórmula.

Sea P una fórmula, si $I(P)$=1 se dice que I satisface a P o que I es un modelo de P [8].

Las interpretaciones de fórmulas se establecen [1] en la siguiente tabla (ver 7):

$I(\text{true})$	=	1
$I(\text{false})$	=	0
$I(\neg P)$	=	$1 \leftrightarrow I(P) = 0$
$I(P \wedge Q)$	=	$1 \leftrightarrow I(P) = I(Q) = 1$
$I(P \vee Q)$	=	$0 \leftrightarrow I(P) = I(Q) = 0$
$I(P \rightarrow Q)$	=	$0 \leftrightarrow I(P) = 1$ e $I(Q) = 0$
$I(P \leftrightarrow Q)$	=	$1 \leftrightarrow I(P) = I(Q)$

Cuadro 7: Tabla de interpretaciones de fórmulas.

Ejemplo 2.6: Si se tiene la fórmula $A = (p \rightarrow q) \rightarrow r$ y la asignación $I_1(p)$=0, $I_1(q) = 0$ e $I_1(r) = 0$ entonces $I_1(A) = 0$ por lo que I_1 no es un modelo para A.

Por otro lado, si se tiene la asignación $I_2(p) = 0$, $I_2(q) = 0$ e $I_2(r) = 1$ entonces $I_2(A) = 1$ por lo que I_2 es un modelo para A.
Se cumplen las siguientes caracterizaciones para cualquier fórmula [8]:

- P es una tautología y se denota como $\models P$ si para toda interpretación I se tiene que $I(P) = 1$.
- P es satisfacible y se denota como $I \models P$ si para alguna interpretación I se tiene que $I(P) = 1$.
- P es insatisfacible y se denota como $I \not\models P$ si para alguna interpretación I se tiene que $I(P) = 0$.
- P es una contradicción si para toda interpretación I se tiene que $I(P)$=0.

3. INSTRUCCIONES

Descarga el archivo PracticaSiete.hs y resuelve los ejercicios definidos sobre éste.

4. EJERCICIOS

Ejercicio 1: crear una instancia adecuada de la clase *Show* para el tipo de dato Prop.

```
1  *Practica2> PTrue
2  True
3  *Practica2> PFalse
4  False
5  *Practica2> Var "a"
6  a
7  *Practica2> Neg (Var "e")
8  ¬e
9  *Practica2>  Conj (Var "p") (Neg (Var "p"))
10 (p ∧ ¬p)
11 *Practica2> Disy PFalse (Var "z")
12 (False ∨ z)
13 *Practica2>  Impl PTrue PTrue
14 (True → True)
15 *Practica2> Syss (Neg PFalse) (Var "s")
16 (¬False ↔ s)
```

Código 1: Ejemplos de ejecución de una nueva instancia de la clase Show para el tipo de dato Prop.

Ejercicio 2: resolver la función *variables* que recibe una expresión de tipo Prop y regresa la lista con todas las variables que aparecen en la expresión.

```
1 variabes :: Prop -> [Nombre]
```

Código 2: Ejemplo de firma de la función *variables*

```
1 *Practica2> variables (Syss (Impl (Var "a") (Var "b")) (Impl (↩
     Var "c") (Conj (Var "b") (Var "c"))))
2 ["a","b","c"]
```

Código 3: Ejemplo de ejecución de la función *variables*.

Ejercicio 3: resolver la función *asociatividad_der* que recibe una expresión de tipo Prop y aplica la ley de la asociatividad hacia la derecha sobre los elementos de la expresión.

```
1 asociatividad_der :: Prop -> Prop
```

Código 4: Ejemplo de firma de la función *asociatividad_der*

```
1 *Practica2> asociatividad_der (Conj (Conj (Var "a") (Var "b"))↩
     (Var "c"))
2 (a ∨ (b ∨ c))
```

```
*Practica2> asociatividad_der (Conj (Var "a") (Conj (Var "b") (Var "c")))
(a ∨ (b ∨ c))
*Practica2> asociatividad_der (Disy (Disy (Var "p") (Var "q")) (Var "r"))
(p ∨ (q ∨ r))
*Practica2> asociatividad_izq (Disy (Var "p") (Disy (Var "q") (Var "r")))
(p ∨ (q ∨ r))
```

Código 5: Ejemplos de ejecución de la función *asociatividad_der*.

Ejercicio 4: resolver la función *asociatividad_izq* que recibe una expresión de tipo Prop y aplica la ley de la asociatividad hacia la izquierda sobre los elementos de la expresión.

```
asociatividad_izq :: Prop -> Prop
```

Código 6: Ejemplo de firma de la función *asociatividad_izq*

```
*Practica2> asociatividad_izq (Conj (Conj (Var "a") (Var "b")) (Var "c"))
((a ∨ b) ∨ c)
*Practica2> asociatividad_izq  (Conj (Var "a") (Conj (Var "b") (Var "c")))
((a ∨ b) ∨ c)
*Practica2> asociatividad_izq  (Disy (Disy (Var "p") (Var "q")) (Var "r"))
((p ∨ q) ∨ r)
*Practica2> asociatividad_izq (Disy (Var "p") (Disy (Var "q") (Var "r")))
((p ∨ q) ∨ r)
```

Código 7: Ejemplos de ejecución de la función *asociatividad_izq*.

Ejercicio 5: resolver la función *conmutatividad* que recibe una expresión de tipo Prop y aplica la ley de la conmutatividad sobre los elementos de la expresión cuyo operador lógico sea conjunción o disyunción.

```
conmutatividad :: Prop -> Prop
```

Código 8: Ejemplo de firma de la función *conmutatividad*

```
*Practica2> conmutatividad (Conj (Neg (Var "u")) PTrue)
(True ∧ ¬u)
*Practica2> conmutatividad (Disy (Var "s") (Var "w"))
(w ∨ s)
```

Código 9: Ejemplos de ejecución de la función *conmutatividad*.

Ejercicio 6: resolver la función *distributividad* que recibe una expresión de tipo Prop y aplica la ley de la distributividad sobre la expresión.

```
1 distributividad :: Prop -> Prop
```

Código 10: Ejemplo de firma de la función *distributividad*

```
1 *Practica2> distributividad (Conj (Neg (Var "d")) (Disy (Var "↩
    e") (Var "f")))
2 ((¬d ∧ e) ∨ (¬d ∧ f))
3 *Practica2> distributividad (Disy (Var "s") (Conj (Var "t") (↩
    Var "u")))
4 ((s ∨ t) ∧ (s ∨ u))
```

Código 11: Ejemplos de ejecución de la función *distributividad*.

Ejercicio 7: resolver la función *deMorgan* que recibe una expresión de tipo Prop y aplica las leyes de De morgan sobre la expresión.

```
1 deMorgan :: Prop -> Prop
```

Código 12: Ejemplo de firma de la función *deMorgan*

```
1 *Practica2> deMorgan (Neg (Conj (Var "a") (Var "b")))
2 (¬a ∨ ¬b)
3 *Practica2> deMorgan (Neg (Disy (Var "c") (Var "d")))
4 (¬c ∧ ¬d)
```

Código 13: Ejemplos de ejecución de la función *deMorgan*.

Ejercicio 8: resolver la función *equiv_op* que recibe una expresión de tipo Prop y aplica la equivalencia de operadores sobre la expresión.

```
1 equiv_op :: Prop -> Prop
```

Código 14: Ejemplo de firma de la función *equiv_op*

```
1 *Practica2> equiv_op (Impl PFalse (Var "q"))
2 (¬False ∨ q)
3 *Practica2> equiv_op (Syss PFalse PTrue)
4 ((¬True ∨ False) ∧ (¬False ∨ True))
```

Código 15: Ejemplos de ejecución de la función *equiv_op*.

Ejercicio 9: resolver la función *dobleNeg* que recibe una expresión de tipo Prop y quita las dobles negaciones existentes sobre la expresión.

```
1 dobleNeg :: Prop -> Prop
```

Código 16: Ejemplo de firma de la función *dobleNeg*

```
*Practica2> dobleNeg (Neg (Neg (Conj (Neg (Neg (Var "x"))) (Var "y"))))
(x ∧ y)
```

Código 17: Ejemplo de ejecución de la función *dobleNeg*.

Ejercicio 10: resolver la función *cuenta_conectivos* que recibe una expresión de tipo Prop y regresa el número de conectivos lógicos en la expresión.

```
cuenta_conectivos :: Prop -> Int
```

Código 18: Ejemplo de firma de la función *cuenta_conectivos*

```
*Practica2> cuenta_conectivos (Syss (Var "n") (Neg (Conj PTrue (Var "m"))))
3
```

Código 19: Ejemplo de ejecución de la función *cuenta_conectivos*.

Ejercicio 11: resolver la función *interpretacion* que recibe una expresión de tipo Prop, un conjunto de asignaciones para las variables y regresa la interpretación de la expresión.

```
interpretacion :: Prop -> Asignacion -> Int
```

Código 20: Ejemplo de firma de la función *interpretacion*

```
*Practica2> interpretacion (Impl (Impl (Var "p") (Var "q")) (Var "r")) [("p",0),("q",0),("r",0)]
0
*Practica2> interpretacion (Impl (Impl (Var "p") (Var "q")) (Var "r")) [("p",1),("q",0),("r",0)]
1
```

Código 21: Ejemplos de ejecución de la función *interpretacion*.

5. ESPECIFICACIONES

✓ Respetar las firmas de las funciones. Cambiarlas podría ser motivo de anulación del ejercicio.

× Cualquier plagio de prácticas será evaluado con 0, sin hacer indagaciones. **Creen** su propio código.

× Cualquier práctica entregada posterior a la fecha límite NO será tomada en cuenta.

Se deberá contar con un difectorio cuyo nombre sea PracticaSeis. Dentro del directorio se deben tener:

- README.txt, donde se incluya nombre y número de cuenta de los integrantes del equipo junto con comentarios que crean pertinentes sobre la práctica.
- PracticaSiete.hs, script requerido para esta práctica.

Compirmir el directorio con el formato **ApellidoNombreP7**. Comprimir con extensión .tar.gz o .zip

Solamente un integrante del equipo deberá enviar la práctica pero deberán enviar el correo con copia a su compañero de trabajo.

Enviar la práctica al correo ciclomax9@ciencias.unam.mx con el asunto `[LC-Apellido-Nombre-P7]`.

"Any fool can write code thtat a computer can understand.

Good programmers write code that humans can understand." - *Martin Fowler*

Suerte ☺

Referencias

[1] Miranda, Favio y Viso, Elisa, *Matemáticas Discretas*. México DF, México: Las prensas de ciencias, 2010.

[2] Rosen, Kenneth H., *Matemáticas Discretas*. Colombia: Concepción Fernández, 2004.

[3] Karl G., Vinfired y Tremblay, Jean-Paul, *Matemática Discreta y lógica: Una perspectiva desde la Ciencia de la Comptuación*. España: Prentice Hall, 1997.

[4] Bustamante A., Alfonso, *Lógica y argumentación: De los argumentos inductivos a las álgebras de Boole*. Colombia: Prentice Hall, 2009.

[5] Pes, Carlos, *Diseño de Algoritmos en Pseudocódigo y Ordinogramas*. CA, USA: Tutorial de Algoritmos de Abrirllave, 2017.

[6] Fernández de C., Max, Preisser, Asunción, Felipe S., Luis y Torres F., Yolanda *Lógica Elemental*. México, 1996.