Introducción y Sintaxis de la Lógica de Proposiciones
Lógica Computacional 2023-I, Nota de clase 1

Favio Ezequiel Miranda Perea Araceli Liliana Reyes Cabello
Lourdes Del Carmen González Huesca Pilar Selene Linares Arévalo

Facultad de Ciencias UNAM

1. Introducción

En muchos aspectos cotidianos hacemos referencia a la lógica, ya sea de forma explícita o implícita pero estricta y formalmente:

¿Qué es la lógica?

- La habilidad para determinar respuestas correctas mediante un proceso estandarizado.
- El estudio de la inferencia formal.
- Una secuencia de afirmaciones verificadas.
- Razonamiento, lo opuesto a la intuición.
- La deducción o derivación de afirmaciones a partir de afirmaciones previas.
- Rama de la filosofía que trata de las formas de razonamiento y pensamiento, especialmente la inferencia y el método científico.
- Estudio de los principios de inferencia válida.

Si bien la lógica es una ciencia muy antigua y la computación de reciente estudio, ambas se han fundido durante el siglo pasado, de manera que hoy en día es difícil marcar una frontera entre ambas ciencias. Entonces

¿Por qué estudiar lógica (computacional)?

- La lógica computacional, que estudiaremos en este curso, refleja el uso de la lógica en computación, en cierto sentido es el *cálculo de la computación*, un fundamento matemático para tratar la información y razonar acerca del comportamiento de programas.
- Además proporciona un entrenamiento que lleva a formas correctas y precisas de razonamiento, en particular permite generar descripciones libres de ambigüedades.
- Es relevante en otras ramas de la computación, en particular para cursos posteriores como inteligencia artificial, bases de datos o ingeniería de software entre otras ya sea de forma explícita o implícita.
- Muchos problemas computacionales involucran lógica en su representación o solución además de ser muy útil en la especificación de software.

- La lógica computacional es una parte fundamental de los métodos formales. Los métodos formales son técnicas para

  especificar al usar notaciones formales como definiciones;

  desarrollar cuando la especificación es parte del diseño, por ejemplo en lenguajes de programacón como la semática operacional o axiomática;

  verificar al demostrar propiedades ya sea "a mano" o con ayuda de la computadora para automatizar, por ejemplo la programación declarativa o los asistentes de prueba.

1.1. Motivación para la formalización del razonamiento correcto

La lógica ha sido pieza clave para estructurar el pensamiento y el razonamiento:

- Dar un fundamento a las matemáticas.
- Eliminar errores del razonamiento.
- Encontrar una forma eficiente para llegar a una justificación de una conclusión, dada cierta información en forma de premisas.

2. Argumentos lógicos

Un *argumento lógico* es una colección finita de afirmaciones (proposiciones) dividida en premisas y conclusión. Las premisas y conclusión deben ser susceptibles de recibir un valor de verdad. El argumento lógico puede ser correcto o incorrecto.

Un problema central de la lógica es verificar la correctud de un argumento lógico. Y una aplicación principal en computación es: *¿cómo saber que el código que nos proporcione un programador es correcto, es decir, realmente el programa siempre cumplirá su especificación?*

Veamos algunos ejemplos de argumentos lógicos:

- *Todos los hombres son mortales, Socrates es hombre. Por lo tanto Socrates es Mortal.* ¿Es correcto?
- *Nada es mejor que Superman, Un taco es mejor que nada. Por lo tanto un taco es mejor que Superman.* ¿Es correcto?
- *Los borogrovos se ponen fefos durante el brilgo. Pac es un borogrovo y hay brilgo. Por lo tanto Pac está fefo.*¿Es correcto?

Pero, ¿cómo decidir si un argumento es correcto? ¿Por sentido común, por votación, por autoridad?

Nosotros usaremos la lógica como un medio para decidir la validez de un argumento: Un argumento es correcto o válido si **suponiendo** que sus premisas son ciertas, entonces necesariamente la conclusión también lo es. Obsérvese que las premisas se suponen siempre ciertas. En el análisis de un argumento lógico no importa el contenido sino la forma de las premisas.

La isla de los caballeros y los bribones En la isla de los caballeros y bribones sólo hay dos clases de habitantes, los caballeros que siempre dicen la verdad y los bribones que siempre mienten. Un náufrago llega a la isla y encuentra dos habitantes: A y B. El habitante A afirma : Yo soy un bribón o B es un caballero.
El acertijo consiste en averiguar cómo son A y B.
Representación lógica:

$p :=$ "A es un bribón", $q :=$ "B es un caballero".
Entonces A dijo $s :=$ p $\vee$ q.

Para resolver el acertijo suponemos que A es bribón.

- En tal caso s es mentira por lo que ambas p y q son falsas.
- Si p es falsa entonces A no es bribón.
- Contradicción: A no puede ser bribón y no bribón a la vez.

De manera que A no es bribón y entonces es caballero y s es cierta. Pero como p es falsa, dado que A es caballero entonces q debe ser cierta de manera que también B es caballero.

Especificación de un sistema de cómputo Considere la siguiente información acerca de un sistema de cómputo:

1. El sistema está en modo multiusuario si y sólo si se comporta normalmente.
2. Si el sistema se comporta normalmente, su núcleo funciona.
3. El núcleo no funciona o el sistema está en modo de interrupción.
4. Si el sistema no está en modo multiusuario entonces está en modo de interrupción.

¿Será posible que bajo estás condiciones el sistema no se encuentre en modo de interrupción? ¿Cómo resolvemos este problema, es decir, cómo tener certeza acerca de esta especificación?

2.1. Características de un argumento lógico

- Los argumentos involucran **individuos** (personas, objetos, cosas en general): *los hombres, Socrates, un taco, Superman, etc..*
- Los individuos tienen **propiedades**: *ser mortal, ser mejor, etc.*
- Una **proposición** es una oración que puede calificarse como verdadera o falsa y que habla de las propiedades de los individuos.
- Los **argumentos** se forman mediante proposiciones, clasificadas como premisas y conclusión del argumento.
- Las proposiciones pueden ser compuestas.
- Un argumento puede ser correcto (válido) o incorrecto (inválido)
- Un argumento nunca es verdadero ni falso.

3. Sistema lógico

3.1. Componentes de un sistema lógico

Cualquier sistema lógico consta al menos de las siguientes tres componentes:

- Sintaxis: lenguaje formal que se utilizará como medio de expresión.
- Semántica: mecanismo que proporciona significado al lenguaje formal dado por la sintaxis.
- Teoría de la prueba: colección de mecanismos puramente sintácticos cuyo propósito fundamental es obtener o identificar las expresiones válidas, respecto a la semántica, de un lenguaje, en particular se encarga de decidir la correctud de un argumento lógico por medios puramtente sintácticos.

3.2. Propiedades de un sistema lógico

Consistencia: se refiere a que en el sistema lógico no hay contradicciones.

Correctud: las reglas del sistema no pueden obtener una inferencia falsa a partir de una verdadera.

Completud: no hay sentencias verdaderas que no se puedan demostrar en el sistema. Es decir, todo lo verdadero es demostrable.

4. Lógica proposicional

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, por ejemplo:

- El es bribón
- Tu eres caballero
- Hoy es jueves.
- Hay vida en Marte.

NO son proposiciones: ¿Llueve?, el perro azul, ¡Viva Pancho Villa!, el cerro de la silla, juan perez, etc.

4.1. El lenguaje PROP

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

- Símbolos o variables proposicionales (un número infinito): $p_1, \ldots, p_n, \ldots$
- 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: $p_1, \ldots, p_n, \ldots$.

- Las constantes $\bot, \top$.

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

1. Si $A \in \mathsf{ATOM}$ entonces $A \in \mathsf{PROP}$. Es decir, toda fórmula atómica es una fórmula.
2. Si $A \in \mathsf{PROP}$ entonces $(\neg A) \in \mathsf{PROP}$
3. Si $A, B \in \mathsf{PROP}$ entonces
   $(A \wedge B), (A \vee B), (A \to B), (A \leftrightarrow B) \in \mathsf{PROP}$.
4. Son todas.

La última cláusula de la definición garantiza que el conjunto PROP es el mínimo conjunto cerrado bajo las tres primeras reglas.

La definición anterior puede darse en la llamada forma de Backus-Naur para definir gramáticas como sigue:

$$\begin{array}{rcl} A, B & ::= & VarP \mid \bot \mid \top \mid (\neg A) \mid (A \wedge B) \mid (A \vee B) \mid (A \to B) \mid (A \leftrightarrow B) \\ VarP & ::= & p_1 \mid p_2 \mid \ldots \mid p_n \mid \ldots \end{array}$$

4.2. Precedencia y Asociatividad de los Operadores Lógicos

Para evitar el uso de paréntesis al máximo definimos la siguiente precedencia de operadores de mayor a menor:

- $\neg$
- $\wedge, \vee$
- $\to$
- $\leftrightarrow$

De manera que la operación a realizar primero es la dada por el conectivo de mayor precedencia. Veamos algunos ejemplos:

$$\begin{array}{rcl} \neg p \to r & \text{es} & (\neg p) \to r \\ p \to q \wedge r & \text{es} & p \to (q \wedge r) \\ \neg p \to q \leftrightarrow r & \text{es} & \big((\neg p) \to q\big) \leftrightarrow r \\ p \to q \wedge r \leftrightarrow s \vee \neg t & \text{es} & \big(p \to q \wedge r\big) \leftrightarrow \big(s \vee (\neg t)\big) \\ p \wedge q \to \neg q \vee s & \text{es} & (p \wedge q) \to ((\neg q) \vee s) \end{array}$$

Obsérvese que los operadores $\wedge, \vee$ tienen la misma precedencia de manera que expresiones como $p \wedge q \vee r$ son ambiguas y no pueden ni deben ser utilizadas.

Acerca de la asociatividad, los operadores $\wedge, \vee, \leftrightarrow$ son asociativos de manera que expresiones como $p \vee q \vee r$ o $\neg p \leftrightarrow p \vee s \leftrightarrow t$ están libres de ambigüedades.
Por otra parte el operador $\to$ no es asociativo pero adoptaremos la convención usual de asociarlo a la derecha, es decir expresiones del estilo $A_1 \to A_2 \to A_3$ se entenderán como $A_1 \to (A_2 \to A_3)$

5. Sintaxis abstracta

Se discutira en clase.

6. Definiciones Recursivas y el Principio de Inducción

Las definiciones recursivas son omnipresentes en Ciencias de la Computación. La definición del lenguaje PROP es un ejemplo de esta clase de definiciones. Una definición recursiva consiste en definir propiedades o funciones de una estructura de datos mediante un análisis de casos, es decir de las distintas formas sintácticas que definen a los elementos de estructura de la datos.

Veamos un ejemplo de definición recursiva. La función $np : \mathsf{PROP} \to \mathbb{N}$ devuelve el número de paréntesis de una fórmula dada y se define como sigue:

$$\begin{array}{rcl} np(A) & = & 0 \quad \text{si } A \text{ es atómica} \\ np((\neg A)) & = & np(A) + 2 \\ np((A \star B)) & = & np(A) + np(B) + 2 \end{array}$$

Si se desean probar propiedades acerca de estructuras o funciones definidas recursivamente como nuestro lenguaje PROP, como por ejemplo que en una fórmula bien construida A, el número de paréntesis $np(A)$ siempre es un número par, lo adecuado es usar el llamado principio de inducción estructural para fórmulas. Este principio lo enunciamos a continuación:

Definición 1 (Principio de Inducción Estructural para PROP) *Sea $\mathcal{P}$ una propiedad acerca de fórmulas del lenguaje PROP. Para probar que toda fórmula $A \in \mathsf{PROP}$ tiene la propiedad $\mathcal{P}$ basta demostrar lo siguiente:*

- *Caso base: toda variable proposicional tiene la propiedad $\mathcal{P}$.*
- *Hipótesis de inducción: suponer que se cumple la propiedad $\mathcal{P}$ para A y B.*
- *Paso inductivo: mostrar, usando la hipótesis de inducción, que*
  1. *$(\neg A)$ cumple $\mathcal{P}$.*
  2. *$(A \star B)$ cumple con $\mathcal{P}$, donde $\star \in \{\to, \wedge, \vee, \leftrightarrow\}$.*

6.1. Algunas funciones de importancia

Las siguientes funciones serán de utilidad más adelante. Su definición recursiva se deja como ejercicio:

- Profundidad de una fórmula: $depth(A)$ devuelve la profundidad o altura del árbol de sintaxis abstracta de A.
- Número de conectivos de una fórmula: $con(A)$ devuelve el número de conectivos de A.
- Variables de una fórmula: $vars(A)$ devuelve el conjunto o lista de variables que figuran en A (sin repeticiones).
- Atómicas en una fórmula: $atom(A)$ devuelve el número de presencias de fórmulas atómicas en A.

Las siguientes relaciones entre algunas de las funciones recién especificadas deben verificarse mediante inducción estructural:

- $con(A) < 2^{depth(A)}$
- $depth(A) \leq con(A)$.
- $atom(A) \leq 2con(A) + 1$.

7. Sustitución

Una operación sintáctica fundamental de las fórmulas proposicionales es la sustitución: en una fórmula dada A, una variable proposicional p cambia por una fórmula B. Así se genera una nueva fórmula denotada $A[p := B]$ obtenida al sustituir todas las presencias de p en A por B. Esta operación se conoce como sustitución textual y se define recursivamente como sigue:

$$\begin{array}{rcl} p[p := B] & = & B \\ q[p := B] & = & q, \text{ si } p \neq q \\ \top[p := B] & = & \top \\ \bot[p := B] & = & \bot \\ (\neg A)[p := B] & = & \neg(A[p := B]) \\ (A \wedge C)[p := B] & = & (A[p := B] \wedge C[p := B]) \\ (A \vee C)[p := B] & = & (A[p := B] \vee C[p := B]) \\ (A \to C)[p := B] & = & (A[p := B] \to C[p := B]) \\ (A \leftrightarrow C)[p := B] & = & (A[p := B] \leftrightarrow C[p := B]) \end{array}$$

Análogamente se define la sustitución simultánea de n variables proposicionales por n fórmulas $A[p_1, \ldots, p_n := B_1, \ldots, B_n]$, también denotada como $A[\vec{p} := \vec{B}\,]$.

La operación de sustitución tiene las siguientes propiedades que deben mostrarse usando inducción estructural:

- Si p no figura en A entonces $A[p := B] = A$.
- Si $p \neq q$ y p no figura en C entonces

$$A[p := B][q := C] = A[q := C][p := B[q := C]]$$

- Si $p \neq q_1, \ldots, q_n$ y p no figura en $B_1, \ldots . B_n$ entonces

$$A[\vec{q}, p := \vec{B}, C] = A[\vec{q} := \vec{B}\,][p := C]$$

Algunos ejemplos correspondientes a las propiedades anteriores son:

- $(p \to q \wedge r)[s := t \vee p] = p \to q \wedge r$, pues s no figura en $p \to q \wedge r$.
- Ejemplificamos ahora la segunda propiedad

$$\begin{array}{rcl} (q \vee r \to t \leftrightarrow s)[r := p \wedge t][t := p \wedge q] & = \\ (q \vee (p \wedge t) \to t \leftrightarrow s)[t := p \wedge q] & = \\ q \vee (p \wedge (p \wedge q)) \to (p \wedge q) \leftrightarrow s \end{array}$$

  Claramente $r \neq t$ y $r \notin Vars(p \wedge q)$ donde $Vars$ es la función que obtiene el conjunto de variables proposicionales de una fórmula, en particular $Vars(p \wedge q) = \{p, q\}$, y tenemos

$$\begin{array}{rcl} (q \vee r \to t \leftrightarrow s)[t := p \wedge q][r := (p \wedge t)[t := p \wedge q]] & = \\ (q \vee r \to t \leftrightarrow s)[t := p \wedge q][r := p \wedge (p \wedge q)] & = \\ (q \vee r \to (p \wedge q) \leftrightarrow s)[r := p \wedge (p \wedge q)] & = \\ q \vee (p \wedge (p \wedge q)) \to (p \wedge q) \leftrightarrow s \end{array}$$

- Un ejemplo de la tercera propiedad es:

$$\begin{aligned}
(r \wedge t \leftrightarrow (q \wedge p \rightarrow s))[r, p, s := p \wedge s, t, r][q := p \vee r] &= \\
((p \wedge s) \wedge t \leftrightarrow (q \wedge t \rightarrow r))[q := p \vee r] &= \\
((p \wedge s) \wedge t \leftrightarrow ((p \vee r) \wedge t \rightarrow r)) &= \\
(r \wedge t \leftrightarrow (q \wedge p \rightarrow s))[r, p, s, q := p \wedge s, t, r, p \vee r]
\end{aligned}$$