# Lógica de Primer Orden: Introducción y Sintaxis
# Lógica Computacional 2018-2, Nota de clase 4

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

Facultad de Ciencias UNAM
6 de marzo de 2018
Material desarrollado bajo el proyecto UNAM-PAPIME PE102117

## 1. Introducción

La lógica ha sido estudiada desde los tiempos de Aristóteles. En sus inicios se dedicó a fundamentar de manera sólida el análisis del razonamiento humano, formalizando las "leyes del pensamiento", es decir, las nociones de argumento válido o inválido. Aristóteles aisló principios lógicos llamados *silogismos* los cuales fueron utilizados para analizar y verificar argumentos hasta la edad media por lógicos como Peter Abelard o Guillermo de Ockham.

En el siglo 19 George Boole y otros desarrollaron versiones algebráicas de la lógica debido a las limitaciones de la teoría del silogismo para manejar relaciones binarias, por ejemplo "x es hijo de y". Esto dio nacimiento a la lógica algebraica, formalismo que aun existe aunque ha perdido popularidad.

Al final del siglo 19 Gottlob Frege desarrollo la lógica cuantificada sentando así los cimientos de gran parte de la lógica moderna, aún cuando Bertrand Russell le mostró que su sistema era inconsistente. Al mismo tiempo Charles S. Pierce de manera independiente, propuso una lógica similar.

La *paradoja de Russell* generó una crisis en los fundamentos de la matemática la cual se resolvío al inicio del siglo XX usando la lógica de primer orden, la cual funge hasta la actualidad como un fundamento de las matemáticas modernas. Debido a su gran poder, la lógica de primer orden también tiene grandes limitaciones que fueron descubiertas en los años 1930 por Kurt Gödel, Alan Turing y Alonzo Church, entre otros. Gödel dio un sistema deductivo completo y correcto para la lógica de primer orden, Alfred Tarski le dio la semántica formal que estudiaremos y que es la base para el estudio de las semánticas denotacionales de los actuales lenguajes de programación y Gerhard Gentzen y otros desarrollarón la teoría de la demostración. A partir de este punto la lógica de primer orden toma su forma actual.

Durante la siguiente parte del curso nos dedicaremos a estudiar los siguientes aspectos de la lógica de primer orden:

- Su sintaxis y semántica formal.
- El proceso de especificación formal del español a la lógica.
- Los métodos para establecer válidez de argumentos:

1. Argumentación semántica (método útil pero prácticamente imposible de implementar de manera eficaz).
2. Tableaux semánticos (método fácilmente implementable)
3. Resolución binaria (método fundamental de la programación lógica)
4. Deducción natural (método completo y correcto, fundamento de los sistemas de tipos para lenguajes de programación).

### 1.1. Lógica proposicional: expresividad vs. decidibilidad

Por un lado, observemos que la lógica proposicional no es lo suficientemente expresiva. Considérese por ejemplo el siguiente razonamiento:

*Algunas personas van al teatro. Todos los que van al teatro se divierten.*
*De manera que algunas personas se divierten.*

La intuición nos dice que el argumento es correcto. Sin embargo la representación correspondiente en lógica proposicional es:

$$p,\ q\ /\therefore\ r \qquad \text{¡Incorrecto!}$$

Otros ejemplos de expresiones que no pueden ser formalizadas adecuadamente en la lógica proposicional son:

*La lista está ordenada.*

*Cualquier empleado tiene un jefe.*

*Si los caballos son animales entonces las cabezas de caballos son cabezas de animales.*

Este problema de expresividad se soluciona al introducir la lógica de predicados.

Por otro lado, la lógica proposicional es decidible, es decir, existen diversos algoritmos para responder a las preguntas

$$\text{¿ } \models \varphi \text{ ?, ¿ Existe } \mathcal{I} \text{ tal que } \mathcal{I} \models \varphi \text{ ?,}$$
$$\text{¿ Es } \Gamma \text{ satisfacible?, ¿ } \Gamma \models \varphi \text{ ?}$$

Como vimos anteriormente, su estudio es de gran importancia en la teoría de complejidad computacional (problema SAT). Esta propiedad, de gran importancia desde el punto de vista computacional, se pierde en la lógica de predicados.

Así, la lógica proposicional nos ofrece decibilidad de algunas propiedades pero nos limita respecto a la expresividad. La lógica de predicados o de primer orden que estudiaremos, nos ofrecerá mayor expresividad para formalizar expresiones más complejas.

Antes de empezar el estudio formal de la lógica de primer orden discutiremos con detalle un ejemplo de especificación no trivial en lógica de proposiciones. De aquí podremos observar una aplicación directa del problema SAT fuera del ámbito de la lógica. Esta clase de representaciones son típicas de los métodos de reducción de problemas en teoría de la complejidad.

## 1.2. Reducción del Sudoku al problema SAT[1]

Consideramos aquí Sudokus que cumplen las siguientes dos propiedades:

1. Tienen una solución única.
2. Pueden resolverse con razonamiento únicamente, es decir, sin búsqueda.

La representación se sirve de 729 variables proposicionales denotadas $s_{xyz}$ con $x, y, z \in \{1, \ldots, 9\}$, cuyo significado es que el número $z$ está en la entrada o celda $xy$. Debe pensarse el tablero de Sudoku como una matriz de manera que la posición $xy$ denota a la celda en el renglón $x$ y columna $y$.
La **especificación** es la siguiente:

- Hay al menos un número en cada celda

$$\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigvee_{z=1}^{9} s_{xyz}$$

  Esta fórmula nos dice que para cada renglón $x$ y para cada columna $y$ hay un número $z$ en la celda $xy$.

- Cada número figura a los más una vez en cada renglón

$$\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{x=1}^{8} \bigwedge_{i=x+1}^{9} (\neg s_{xyz} \vee \neg s_{iyz})$$

  Esta fórmula nos dice que para cada columna $y$, para cada número $z$ y para cada renglón $x$, excepto el último, si el número $z$ está en el renglón $x$ (en la celda $xy$) entonces no sucede que el número $z$ está en alguno de los renglones posteriores a $x$ (los denotados por $i$).

- Cada número figura a lo más una vez en cada columna

$$\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{y=1}^{8} \bigwedge_{i=y+1}^{9} (\neg s_{xyz} \vee \neg s_{xiz})$$

  Esta fórmula nos dice que para cada renglón $x$, para cada número $z$ y para cada columna $y$, excepto la última, si el número $z$ está en la columna $y$ (en la celda $xy$) entonces no sucede que el número $z$ está en alguna de las columnas posteriores a $y$ (las denotadas por $i$).

- Cada número figura a lo más una vez en cada cuadrante de $3 \times 3$:

  - Dentro de cada cuadrante, si un número $z$ está en un renglón entonces no está en las celdas siguientes en el mismo renglón. Aquí $i, j$ delimitan las celdas válidas del cuadrante y $k$ se encarga de verificar las columnas siguientes a la de la celda en consideración. Todo dentro del mismo cuadrante.

$$\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=y+1}^{3} \left(\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+x)(3j+k)z}\right)$$

[1] Esta sección se basa en el artículo *Sudoku as a SAT Problem* de Inês Lynce y Joël Ouaknine. 9th International Symposium on Artificial Intelligence and Mathematics, January 2006.

- Dentro de cada cuadrante, si un número $z$ está en una columna entonces no está en los renglones superiores ni en la misma columna ni en las columnas siguientes. Aquí $i, j$ delimitan las celdas válidas del cuadrante, $k$ se encarga de verificar los renglones superiores y $l$ verifica la columna actual y las columnas siguientes.

$$\bigwedge_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3}\bigwedge_{k=x+1}^{3}\bigwedge_{l=1}^{3}(\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+k)(3j+l)z})$$

Las cláusulas anteriores conforman una codificación mínima. Un bonito cálculo combinatorio lleva a concluir que a partir de las fórmulas anteriores la fórmula resultante de la transformación a FNC tendrá 8829 cláusulas de las cuales 81 cláusulas tienen 9 literales, y las restantes 8748 son binarias, es decir, tienen 2 literales.

Adicionalmente se deben considerar las cláusulas unitarias correspondientes a las entradas del Sudoku que ya tienen un número asignado. Al agregar las siguientes cláusulas modelamos la **unicidad** de un número en cada celda:

- Hay a lo más un numero en cada celda.

$$\bigwedge_{x=1}^{9}\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{8}\bigwedge_{i=z+1}^{9}(\neg s_{xyz} \vee \neg s_{xyi})$$

  Esta fórmula nos dice que para cada renglón $x$, para cada columna $y$, para cada número no 9, si la celda $xy$ tiene al numero $z$ entonces no tiene a los números mayores que $z$.

- Cada número figura al menos una vez en cada columna

$$\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{9}\bigvee_{x=1}^{9} s_{xyz}$$

  Está fórmula nos dice que para cada columna $y$ y cada número $z$, existe un renglón $x$, tal que $z$ está en la celda $xy$.

- Cada número figura al menos una vez en cada renglón

$$\bigwedge_{x=1}^{9}\bigwedge_{z=1}^{9}\bigvee_{y=1}^{9} s_{xyz}$$

  Está fórmula nos dice que para cada renglón $x$ y cada número $z$, existe una columna $y$, tal que $z$ está en la celda $xy$.

- Cada número figura al menos una vez en cada cuadrante de $3 \times 3$.

$$\bigvee_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3} s_{(3i+x)(3j+y)z}$$

  Está fórmula nos dice que hay un número $z$ en cada uno de los cuadrantes, los cuales qse generan por las celdas $(3i + x)(3j + y)$.

La fórmula resultante de transformar todas las anteriores consta de 11988 cláusulas, 324 de ellas con 9 literales y las restantes 11664 binarias. Nuevamente debemos agregar cláusulas unitarias para las entradas previamente asignadas del Sudoku.

De los cálculos anteriores podemos ver que a pesar de ser un problema pesado, la implementación de la teoría de prueba resolución binaria puede hacer a la computadora una herramienta útil para resolver sudokus.

# 2. Lógica de predicados de manera informal

En la lógica proposicional debemos representar a ciertas frases del español como por ejemplo

*Algunos programas usan ciclos anidados*

mediante fórmulas proposicionales atómicas, es decir mediante una simple variable proposicional $p$. En la lógica de predicados descomponemos a este tipo de frases distinguiendo dos categorias: objetos y relaciones entre objetos. Ambas categorias construidas sobre un universo de discurso fijo el cual varía según el problema particular que se desee representar. Estudiemos con mas detalle estas categorias:

## 2.1. Objetos y universo de discurso

*Algunas **personas** van al teatro*
*Todos los **programas** en Java usan clases.*

- Universo o dominio de discurso: conjunto de objetos que se consideran en el razonamiento, pueden ser cosas, personas, datos, objetos abstractos, etc.
- En el caso de los enunciados anteriores en el universo podemos tener personas y programas en Java, y posiblemente el teatro y las clases.

## 2.2. Relaciones entre objetos

- Las propiedades o relaciones atribuibles a los objetos del dominio de discurso se representan con predicados.
- En el caso de los enunciados anteriores tenemos *ir al teatro, usar clases.*

## 2.3. Cuantificadores

Los cuantificadores especifican el número de individuos que cumplen con el predicado. Por ejemplo:

- *Todos* los estudiantes trabajan duro.
- *Algunos* estudiantes se duermen en clase.
- *La mayoría* de los maestros están locos
- *Ocho de cada diez* gatos lo prefieren.
- *Nadie* es más tonto que yo.
- *Al menos seis* estudiantes están despiertos.
- *Hay una infinidad* de números primos.
- *Hay más* PCs que Macs.

En lógica de primer orden sólo consideraremos dos cuantificadores: *todos* y *algunos.*

### 2.4. Proposiciones vs. Predicados

El uso de predicados en lugar de proposiciones podría parecer simplemente otra manera de escribir las cosas, por ejemplo, la proposición:

*Chubaka recita poesía nordica*

se representa con predicados como

*Recita(Chubaka, poesía nordica)*

La gran ventaja es que el predicado puede cambiar de argumentos, por ejemplo

*Recita(Licantro, odas en sánscrito)*

o en el caso general podemos usar variables para denotar individuos:

*Recita(x,y)*

Obsérvese que está última expresión no es una proposición. En este sentido un mismo predicado está representando un número potencialmente infinito de proposiciones.

Reconozcamos ahora las componentes anteriores en un argumento particular:

*Todos los matemáticos estudian algún teorema. Los teoremas son elementales o trascendentes. Giuseppe es matemático. Luego entonces, Giuseppe estudia algo elemental o trascendente.*

- Universo: *matemáticos y teoremas*
- Predicados: *estudiar, elemental, trascendente.*
- Cuantificadores: *todos, los, algo.*
- Los individuos se representarán formalmente mediante dos categorías:
  - Constantes: las cuales denotan individuos particulares, como Giuseppe.
  - Variables: las cuales denotan individuos genéricos o indeterminados como los matemáticos o el teorema.
    **Obsérvese que** las variables no figuran en el lenguaje natural como el español, pero son necesarias para la formalización y representación de individuos no fijos.

Formalicemos ahora todos los conceptos discutidos anteriormente.

## 3. Sintaxis de la lógica de primer orden

En contraposición al lenguaje de la lógica proposicional que está determinado de manera única, no es posible hablar de un solo lenguaje para la lógica de predicados. Dependiendo de la estructura semántica que tengamos en mente será necesario agregar símbolos particulares para denotar objetos y relaciones entre objetos. De esta manera el alfabeto consta de dos partes ajenas entre sí, la parte común a todos los lenguajes está determinada por los símbolos lógicos y auxiliares y la parte particular, llamada tipo de semejanza o signatura del lenguaje. La parte común a todos los lenguajes consta de:

- Un conjunto infinito de variables $\mathsf{Var} = \{x_1, \ldots, x_n, \ldots\}$

- Constantes lógicas: $\bot, \top$
- Conectivos u operadores lógicos: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- Cuantificadores: $\forall, \exists$.
- Símbolos auxiliares: ( , ) y , (coma).
- Si se agrega el símbolo de igualdad $=$, decimos que el lenguaje tiene igualdad.

La signatura de un lenguaje en particular está dada por:

- Un conjunto $\mathcal{P}$, posiblemente vacío, de símbolos o letras de predicado: $P_1, \ldots, P_n, \ldots$
  A cada símbolo se le asigna un índice [2] o número de argumentos $m$, el cual se hace explícito escribiendo $P_n^{(m)}$ que significa que el símbolo $P_n$ necesita de $m$ argumentos.
- Un conjunto $\mathcal{F}$, posiblemente vacío, de símbolos o letras de función: $f_1, \ldots, f_n, \ldots$
  Análogamente a los símbolos de predicado cada símbolo de función tiene un índice asignado, $f_n^{(m)}$ significará que el símbolo $f_n$ necesita de $m$ argumentos.
- Un conjunto $\mathcal{C}$, posiblemente vacío, de símbolos de constante: $c_1, \ldots, c_n, \ldots$
  En algunos libros los símbolos de constante se consideran como parte del conjunto de símbolos de función, puesto que pueden verse como funciones de índice cero, es decir, funciones sin argumentos.

Dado que un lenguaje de primer orden queda determinado de manera única por su signatura, abusaremos de la notación y escribiremos $\mathcal{L} = \mathcal{P} \cup \mathcal{F} \cup \mathcal{C}$ para denotar al lenguaje dado por tal signatura.

### 3.1. Términos

Los términos del lenguaje son aquellas expresiones que representarán objetos en la semántica y su definición es:

- Los símbolos de constante $c_1, \ldots, c_n, \ldots$ son términos.
- Las variables $x_1, \ldots, x_n, \ldots$ son términos.
- Si $f^{(m)}$ es un símbolo de función y $t_1, \ldots, t_m$ son términos entonces $f(t_1, \ldots, t_m)$ es un término.
- Son todos.

Es decir, los términos están dados por la siguiente gramática:

$$t ::= x \mid c \mid f(t_1, \ldots, t_m)$$

Y el conjunto de términos de un lenguaje dado se denota con $\mathsf{TERM}_{\mathcal{L}}$, o simplemente $\mathsf{TERM}$ si es claro cuál es el lenguaje.

En algunas ocasiones se encuentra una definición de términos que no incluye constantes. Esta omisión no existe en realidad puesto que las constantes se consideran casos particulares de símbolos de función de índice cero como hemos mencionado antes, es decir un símbolo de función que no recibe argumentos. Considerando lo anterior, presentamos la siguiente implementación.

[2] Este índice suele llamarse también "aridad", pero dado que tal palabra no existe en el diccionario de la lengua española, trataremos de evitar su uso.

### 3.1.1. Implementación

Emplearemos la siguiente representación en HASKELL:

```
type Nombre = String

data Term = V Nombre | F Nombre [Term]
```

Veamos algunos ejemplos:

- $c$ se implementa como `F ‘‘c’’ [ ]`
- $f(x, y)$ se implementa como `F ‘‘f’’ [V ‘‘x’’, V ‘‘y’’]`
- $g(a)$ se implementa como `F ‘‘g’’ [F ‘‘a’’ []]`
- $h(f(x))$ se implementa como `F ‘‘h’’ [F ‘‘f’’ [V ‘‘x’’]]}`
- $h(b, f(a), z)$ se implementa como `‘‘h’’ [F ‘‘b’’ [], F ‘‘f’’ [F ‘‘a’’ []], V ‘‘z’’]`

## 3.2. Fórmulas

El siguiente paso es determinar las expresiones o fórmulas atómicas, dadas por:

- Las constantes lógicas $\bot, \top$.
- Las expresiones de la forma: $P_1(t_1, \ldots, t_n)$ donde $t_1, \ldots, t_n$ son términos
- Las expresiones de la forma $t_1 = t_2$, si el lenguaje cuenta con igualdad

El conjunto de expresiones atómicas se denotará con $\mathsf{ATOM}_{\mathcal{L}}$.

Así el conjunto $\mathsf{FORM}_{\mathcal{L}}$ de expresiones compuestas aceptadas en un lenguaje $\mathcal{L}$, llamadas usualmente fórmulas, se define recursivamente como sigue:

- Si $\varphi \in \mathsf{ATOM}_{\mathcal{L}}$ entonces $\varphi \in \mathsf{FORM}_{\mathcal{L}}$. Es decir, toda fórmula atómica es una fórmula.
- Si $\varphi \in \mathsf{FORM}_{\mathcal{L}}$ entonces $(\neg\varphi) \in \mathsf{FORM}_{\mathcal{L}}$.
- Si $\varphi, \psi \in \mathsf{FORM}_{\mathcal{L}}$ entonces $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \to \psi), (\varphi \leftrightarrow \psi) \in \mathsf{FORM}_{\mathcal{L}}$.
- Si $\varphi \in \mathsf{FORM}_{\mathcal{L}}$ y $x \in \mathsf{Var}$ entonces $(\forall x\varphi), (\exists x\varphi) \in \mathsf{FORM}_{\mathcal{L}}$.
- Son todas.

La gramática para las fórmulas en forma de Backus-Naur es:

$$\begin{array}{rcl} \varphi & ::= & at \mid (\neg\varphi) \mid (\varphi \wedge \psi) \mid (\varphi \vee \psi) \mid (\varphi \to \psi) \mid (\varphi \leftrightarrow \psi) \mid (\forall x\varphi) \mid (\exists x\varphi) \\ at & ::= & \bot \mid \top \mid P(t_1, \ldots, t_m) \mid t_1 = t_2 \end{array}$$

Cada lenguaje definido a partir de una signatura y con forme a la gramática recién descrita será un *lenguaje de primer orden.*
Las siguientes observaciones son de suma importancia:

- Los términos y las fórmulas son dos categorías sintácticas ajenas, es decir **ningún** término es fórmula y **ninguna** fórmula es término.
- Los términos denotan exclusivamente individuos u objetos.
- Las fórmulas atómicas denotan proposiciones o propiedades acerca de los términos.
- **Sólo** los individuos u objetos son cuantificables y **únicamente** sobre ellos se puede "predicar". Esta característica justifica la denominación "primer orden".

### 3.3. Precedencia

Si bien los cuantificadores no son propiamente operadores entre fórmulas y por lo tanto no se puede hablar de una precedencia en el sentido usual, usamos la siguiente convención:

Los cuantificadores se aplican a la *mínima* expresión sintácticamente posible.

de manera que

$$\begin{array}{rcl} \forall x\varphi \to \psi & \text{es} & (\forall x\varphi) \to \psi \\ \exists y\varphi \wedge \forall w\psi \to \chi & \text{es} & (\exists y\varphi) \wedge (\forall w\psi) \to \chi \end{array}$$

### 3.4. Implementación

La implementación de fórmulas es la siguiente:

```
data Form = TrueF | FalseF | Pr Nombre [Term] | Eq Term Term |
            Neg Form | Conj Form Form | Disy Form Form | Imp Form Form |
            Equi Form Form | All Nombre Form | Ex Nombre Form
```

Algunos ejemplos son:

- $P(x, y)$ se implementa como `Pr ‘‘P’’ [V ‘‘x’’, V ‘‘y’’]`
- $Q(a, h(w))$ se implementa como `Pr ‘‘Q’’ [F ‘‘a’’ [], F ‘‘h’’ [V ‘‘w’’]]`
- $R(x) \to x = b$ se implementa como `Imp (Pr ‘‘R’’ [V ‘‘x’’]) (Eq (V ‘‘x’’) (F ‘‘b’’ []))`
- $\neg R(z) \vee (S(a) \wedge f(x) = z)$ se implementa como

```
Disy (Neg (Pr ‘‘R’’ [V ‘‘z’’]))
     (Conj (Pr ‘‘S’’ [F ‘‘a’’ []])
           (Eq (F ‘‘f’’ [V ‘‘x’’]) (V ‘‘z’’))
     )
```

- $\forall x Q(x, g(x))$ se implementa como `All ‘‘x’’ (Pr ‘‘Q’’ [V ‘‘x’’,F ‘‘g’’ [V ‘‘x’’]])`
- $\forall x \exists y P(x, y)$ se implementa como `All ‘‘x’’ (Ex ‘‘y’’ (Pr ‘‘P’’ [V ‘‘x’’,V ‘‘y’’]))`

# 4. Inducción y Recursión

Las definiciones recursivas para los elementos del lenguaje generan principios de inducción. Dado que ahora tenemos dos categorías sintácticas, términos y fórmulas, tenemos también principios de inducción correspondientes a cada categoría.

**Definición 1 (Principio de Inducción Estructural para $\mathsf{TERM}_{\mathcal{L}}$)** *Sea $\mathcal{P}$ una propiedad acerca de términos. Para demostrar que $\mathcal{P}$ es válida para todos los términos, basta seguir los siguientes pasos:*

- *Base de la inducción: mostrar que*
  1. *Si $x \in \mathsf{Var}$ entonces $\mathcal{P}$ es válida para $x$.*
  2. *Si $c \in \mathcal{C}$ entonces $\mathcal{P}$ es válida para $c$.*
- *Hipótesis de inducción: suponer $\mathcal{P}$ para cualesquiera $t_1, \ldots, t_n \in \mathsf{TERM}_{\mathcal{L}}$.*
- *Paso inductivo: usando la Hipótesis de inducción mostrar que*
  1. *Si $f \in \mathcal{F}$ es un símbolo de función de índice $n$ entonces $f(t_1, \ldots, t_n)$ cumple $\mathcal{P}$.*

**Definición 2 (Principio de Inducción Estructural para $\mathsf{FORM}_{\mathcal{L}}$)** *Sea $\mathcal{P}$ una propiedad acerca de fórmulas. Para probar que toda fórmula $\varphi \in \mathsf{FORM}_{\mathcal{L}}$ tiene la propiedad $\mathcal{P}$ basta seguir los siguientes pasos:*

- *Caso base: mostrar que toda fórmula atómica tiene la propiedad $\mathcal{P}$.*
- *Hipótesis de inducción: suponer que $\varphi$ y $\psi$ cumplen $\mathcal{P}$.*
- *Paso inductivo: mostrar usando la Hipótesis de inducción que*
  1. *$(\neg\varphi)$ también cumple $\mathcal{P}$.*
  2. *$(\varphi \star \psi)$ tiene la propiedad $\mathcal{P}$, donde $\star \in \{\rightarrow, \wedge, \vee, \leftrightarrow\}$*
  3. *$\forall x\varphi$ y $\exists x\varphi$ cumplen $\mathcal{P}$.*

Con respecto a la recursión, también podemos definir funciones sobre los términos y las fórmulas usando los siguientes principios:

**Definición Recursiva para Términos** Para definir una función $h : \mathsf{TERM}_{\mathcal{L}} \rightarrow A$, basta definir $h$ como sigue:

- Definir $h(x)$ para $x \in \mathsf{Var}$.
- Definir $h(c)$ para cada constante $c \in \mathcal{C}$.
- Suponiendo que $h(t_1), \ldots, h(t_n)$ están definidas, definir $h(f(t_1, \ldots, t_n))$ para cada símbolo de función $f \in \mathcal{F}$ de índice $n$, utilizando $h(t_1), \ldots, h(t_n)$.

**Definición Recursiva para Fórmulas** . Para definir una función $h : \mathsf{FORM}_{\mathcal{L}} \to A$, basta definir $h$ como sigue:

- Definir $h$ para cada fórmula atómica, es decir, definir $h(\bot), h(\top), h(P(t_1, \ldots, t_n))$ y $h(t_1 = t_2)$ si el lenguaje tiene igualdad.
- Suponiendo definidas $h(\varphi)$ y $h(\psi)$, definir a partir de ellas a $h(\neg\varphi),\ h(\varphi \vee \psi),\ h(\varphi \wedge \psi),\ h(\varphi \to \psi),\ h(\varphi \leftrightarrow \psi),\ h(\forall x\varphi)$ y $h(\exists x\varphi)$.

Veamos algunos ejemplos:

**Ejemplo 4.1** Definimos el conjunto de subtérminos de un término $t$ recursivamente como sigue:

- $\mathsf{Subt}(x) = \{x\}$
- $\mathsf{Subt}(c) = \{c\}$
- $\mathsf{Subt}(f(t_1, \ldots, t_n)) = \{f(t_1, \ldots, t_n)\} \cup \mathsf{Subt}(t_1) \cup \ldots \cup \mathsf{Subt}(t_n)$

**Ejemplo 4.2** Definimos recursivamente el número de conectivos y cuantificadores de una fórmula, conocido como el peso de una fórmula, como sigue:

- $\mathsf{peso}(\bot) = \mathsf{peso}(\top) = 0$.
- $\mathsf{peso}(P(t_1, \ldots, t_n)) = 0$.
- $\mathsf{peso}(\neg\varphi) = \mathsf{peso}(\varphi) + 1$.
- $\mathsf{peso}(\varphi \wedge \psi) = \mathsf{peso}(\varphi) + \mathsf{peso}(\psi) + 1$
- $\mathsf{peso}(\forall x\varphi) = \mathsf{peso}(\varphi) + 1$