

# Lógica Computacional 2022-1, nota de clase 12

## Fundamentos de programación lógica

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

17 de diciembre de 2021

### 1. Breve introducción a la programación lógica

Los lenguajes de programación más ampliamente usados, como C o JAVA, forman parte del paradigma de programación imperativa o procedimental cuyas características principales son:

- Un programa es una secuencia de instrucciones.
- La principal estructura de control son los ciclos: *while*, *repeat*, *for* etc.
- La operación de asignación  $x := a$  es imprescindible.
- Las estructuras de control permiten seguir paso a paso las acciones que debe realizar un programa.
- Es decir, el programa especifica *cómo* se calculan los resultados.

En contraste los lenguajes de programación funcional (HASKELL, LISP, SCHEME, ML) y lógica (PROLOG ) conforman la llamada programación declarativa cuyas características principales son:

- Un programa es una sucesión de definiciones.
- La principal estructura de control es la recursión.
- No existen ni ciclos ni operación de asignación
- El programa especifica *qué* se debe calcular, es decir, las propiedades que debe cumplir el resultado o solución a calcular.
- El *cómo* es irrelevante.

#### 1.1. Ventajas de la programación declarativa

La programación declarativa no depende del lenguaje en particular.

- Si bien los programas imperativos pueden ser rápidos y especializados, un programa declarativo es más general, corto y legible.
- Debido a lo anterior podemos decir que los programas declarativos son elegantes matemáticamente. Lo cual implica que es más fácil *verificar* si el programa cumple su especificación.
- Aprender programación declarativa permite al programador desarrollar un estilo de programación riguroso y disciplinado que puede ser usado ventajosamente sin importar el lenguaje de programación utilizado.
- Este estilo genera programas con una mejor ingeniería, más fáciles de depurar, mantener y modificar.

## 1.2. Lenguajes de programación lógica

Fue alrededor de las décadas de 1920 y 1930 que Jacques Herbrand <sup>1</sup> propuso en su tesis un método para verificar la validez de fórmulas en lógica de predicados utilizando un procedimiento para unificar fórmulas. Esta tesis es el fundamento de la programación lógica que modela al cómputo a través de los llamados modelos de Herbrand donde:

1. El dominio es el universo formado por términos que representan objetos
2. Las fórmulas que involucran predicados describen un problema

Las fórmulas son usadas para expresar conocimiento, en particular, descripciones o algoritmos en forma de funciones parciales creando un programa lógico.

Realizar cálculos a partir de un programa lógico es obtener respuestas a partir de la información descrita. Las respuestas también llamadas metas son exactamente las sustituciones que asocian valores del universo de Herbrand a las variables de la meta.

Así, el significado (declarativo) de un programa y sus respuestas están bien definidas a través de un modelo matemático que ofrece el universo de Herbrand.

Lo anterior permite establecer las características de un lenguaje de programación lógica: un lenguaje declarativo en el que los programas constan de definiciones plasmadas en fórmulas; en particular los predicados *especifican* información acerca de lo que se desea calcular, expresada mediante ciertos hechos y reglas, es decir, al establecer relaciones que describan propiedades de la información.

La evaluación de un programa en un lenguaje de programación lógica es *interactiva*: para activar el mecanismo de ejecución se necesita de una pregunta relacionada con la información dada en el programa, es decir, el programa  $\mathbb{P}$  se activa al *preguntar* cierta información  $\mathcal{C}$  lo cual formalmente requiere *verificar* si  $\mathbb{P} \models \mathcal{C}$  ( $\mathcal{C}$  es consecuencia<sup>2</sup> lógica de  $\mathbb{P}$ ).

Los fundamentos del paradigma de programación lógica se sirven básicamente de la lógica de primer orden, en particular el mecanismo de ejecución se basa en la regla de resolución binaria con unificación de Robinson. Esta regla, propuesta alrededor de 1960 por Alan Robinson, es esencialmente la presentada en la nota 10. Robinson también introdujo el primer algoritmo de unificación cuya complejidad es mayor a la del algoritmo que nosotros estudiamos y utilizamos (el de Martelli y Montanari descrito en la nota 9).

El poder expresivo de la lógica inspirado en la tesis de Herbrand y el proceso refinado para la resolución de Robinson, fueron usados en combinación por Robert Kowalski, Alan Colmerauer y Philippe Roussel alrededor de 1970 para crear el primer lenguaje de programación lógica: PROLOG .

## 2. Resolución Binaria en Programación Lógica

### 2.1. Notaciones para cláusulas

Además de la notación de cláusulas utilizada hasta ahora, se pueden utilizar otras, aquí mencionamos dos de ellas.

---

<sup>1</sup>Jacques Herbrand (1908-1931), prominente lógico que murió a la edad de 23 años en un accidente en los alpes franceses.

<sup>2</sup>La definición de este concepto semántico es la misma que en lógica de predicados: una fórmula  $A$  es consecuencia lógica de un conjunto de fórmulas  $\Gamma$  si y sólo si todo modelo de  $\Gamma$  es modelo de  $A$ . En este caso el conjunto particular de fórmulas es un programa lógico  $\mathbb{P}$  y la fórmula consultada es una cláusula  $\mathcal{C}$

**Notación conjuntista** Una notación para cláusulas utilizada por diversos autores es la notación conjuntista que representa a la cláusula  $\mathcal{C} = \ell_1 \vee \ell_2 \vee \dots \vee \ell_n$  como el conjunto

$$\mathcal{C} = \{\ell_1, \ell_2, \dots, \ell_n\}$$

La justificación de esta notación está en el hecho de que el orden de las literales no importa, ni tampoco el hecho de que haya literales repetidas. Esta notación **no** será utilizada en nuestro curso.

**Notación para programación lógica** La programación lógica utiliza cláusulas escritas de una manera particular que ahora describimos.

Consideremos una cláusula  $\mathcal{C}$  de la forma

$$\mathcal{C} = \ell_1 \vee \ell_2 \vee \dots \vee \ell_n$$

mediante conmutatividad del  $\vee$  podemos reescribir a  $\mathcal{C}$  de la siguiente forma, agrupando las literales positivas y negativas:

$$\neg P_1 \vee \neg P_2 \vee \dots \vee \neg P_j \vee Q_1 \vee Q_2 \vee \dots \vee Q_k$$

donde  $j + k = n$ , los  $P_i$  y los  $Q_l$  son predicados, llamados átomos en programación lógica (por el momento omitimos los argumentos de cada átomo pues no son relevantes).

Ahora mediante las leyes de De Morgan podemos hacer la siguiente transformación:

$$\neg(P_1 \wedge P_2 \wedge \dots \wedge P_j) \vee (Q_1 \vee Q_2 \vee \dots \vee Q_k)$$

Enseguida podemos escribir esta expresión como una implicación:

$$P_1 \wedge P_2 \wedge \dots \wedge P_j \rightarrow Q_1 \vee Q_2 \vee \dots \vee Q_k$$

Convenimos en sustituir las conjunciones y las disyunciones del consecuente con comas. Siempre debe recordarse que las comas del antecedente representan conjunciones mientras que las del consecuente representan disyunciones.

$$P_1, P_2, \dots, P_j \rightarrow Q_1, Q_2, \dots, Q_k$$

Finalmente convenimos en escribir la implicación al revés obteniendo:

$$Q_1, Q_2, \dots, Q_k \leftarrow P_1, P_2, \dots, P_j$$

que es la presentación de  $\mathcal{C}$  en notación de programación lógica.

En tal caso los átomos  $Q_1, Q_2, \dots, Q_k$  forman la *cabeza* de la cláusula  $\mathcal{C}$  y los átomos  $P_1, P_2, \dots, P_j$  constituyen el *cuerpo* de la cláusula  $\mathcal{C}$ .

## 2.2. Clasificación de cláusulas

Las diversas formas de una cláusula  $\mathcal{C} = Q_1, Q_2, \dots, Q_k \leftarrow P_1, P_2, \dots, P_j$ . según sean  $k$  y  $j$  son las siguientes:

1. Si  $k = 1$  y  $j = 0$ , entonces  $\mathcal{C}$  es de la forma

$$Q_1 \leftarrow$$

en este caso el cuerpo está vacío y la cabeza es un átomo. Tal cláusula se conoce como *hecho*.

2. Si  $k \geq 1$  y  $j = 0$ , entonces  $\mathcal{C}$  es de la forma

$$Q_1, \dots, Q_k \leftarrow$$

El cuerpo está vacío y la cabeza es una disyunción de átomos. Esta cláusula se llama *cláusula positiva*.

3. Si  $k = 1$  y  $j \geq 0$ , entonces  $\mathcal{C}$  es de la forma

$$Q_1 \leftarrow P_1, \dots, P_j$$

Esta cláusula se llama *cláusula de Horn*, *cláusula definida* o *regla*.

4. Si  $k > 1$  y  $j \geq 0$ , entonces  $\mathcal{C}$  es de la forma

$$Q_1, Q_2, \dots, Q_k \leftarrow P_1, P_2, \dots, P_j$$

Este es el caso general y se llama *cláusula disyuntiva* o cláusula que no es de Horn.

5. Si  $k = 0$  y  $j \geq 1$ , entonces  $\mathcal{C}$  es de la forma

$$\leftarrow P_1, \dots, P_j$$

La cabeza esta vacía. Este tipo de cláusula se llama *meta*, *objetivo* o *cláusula negativa*.

6. Si  $k = 0$  y  $j = 0$ , entonces  $\mathcal{C}$  es de la forma

$$\leftarrow$$

Esta cláusula representa a la *cláusula vacía*.

Veamos algunos ejemplos de especificaciones escrita en forma clausular para la programación lógica. Es importante recalcar que la lógica de la programación lógica corresponde a la lógica de predicados **sin** igualdad por lo que debemos trabajar con especificaciones relacionales (véase la sección 2 de la nota 6). Esto no quiere decir que no podamos usar símbolos de función, sino que no podemos usar el símbolo de igualdad  $=$  para definir funciones.

**Ejemplo 1** La especificación de la estructura de datos para números naturales, mediante un predicado *Nat* es:

- $\text{Nat } 0$
- $\forall x (\text{Nat } x \rightarrow \text{Nat } sx)$

donde  $0$  es el símbolo de constante para el número cero y  $s$  es el símbolo de función para la función sucesor. Las cláusulas correspondientes son:

- $\text{Nat } 0 \leftarrow$
- $\text{Nat } sx \leftarrow \text{Nat } x$

**Ejemplo 2** La especificación de la función suma de números naturales en forma relacional (es decir con predicados y no con funciones) es:

- $\forall x P x 0 x$
- $\forall x \forall y \forall z (P x y z \rightarrow P x sysz)$

Las cláusulas correspondientes son:

- $P x 0 x \leftarrow$
- $P x sysz \leftarrow P x y z$

**Ejemplo 3** La especificación de la función producto de números naturales en forma relacional (es decir con predicados y no con funciones) es:

- $\forall x M0x0$
- $\forall x \forall y \forall z \forall v (Mxyv \wedge Pvyz \rightarrow Msxyz)$

Se observa el uso auxiliar del predicado suma  $P$ . Las cláusulas correspondientes son:

- $M0x0 \leftarrow$
- $Msxyz \leftarrow Mxyv, Pvyz$

**Ejemplo 4** La especificación de la relación de orden  $<$  en números naturales es:

- $\forall x L0sx$
- $\forall x \forall y (Lxy \rightarrow Lxsy)$

Las cláusulas correspondientes son:

- $L0sx \leftarrow$
- $Lxsy \leftarrow Lxy$

## 2.3. Resolución binaria y programas lógicos

Con la nueva notación la regla de resolución binaria se escribe como sigue:

$$\frac{\begin{array}{l} \mathcal{C} = Q_1, \dots, Q_k, \ell \leftarrow P_1, \dots, P_j \\ \mathcal{D} = S_1, \dots, S_m \leftarrow \ell', R_1, \dots, R_n \\ \mu \text{ un umg de } \{\ell, \ell'\} \quad \text{Var}(\mathcal{C}) \cap \text{Var}(\mathcal{D}) = \emptyset \end{array}}{(Q_1, \dots, Q_k, S_1, \dots, S_m \leftarrow P_1, \dots, P_j, R_1, \dots, R_n)\mu}$$

**Definición 1** Un programa lógico  $\mathbb{P}$  es un conjunto finito de cláusulas no negativas.

De ahora en adelante nos interesa hacer resolución principalmente sobre programas lógicos definidos, que son aquellos que el lenguaje de programación PROLOG permite implementar.

**Definición 2** Un programa lógico definido  $\mathbb{P}$  es un conjunto finito de cláusulas de Horn, es decir, un conjunto finito de hechos y reglas.

Omitiremos el adjetivo “definido” pues no trataremos con otro tipo de programas. Obsérvese que las metas nunca son parte de un programa lógico, sino que sirven para interactuar con un programa mediante un intérprete que se encarga de buscar la cláusula vacía mediante resolución binaria. Veamos un par de ejemplos.

**Ejemplo 2.1** Considérese el programa para la suma de naturales

$$\mathbb{P}_+ = \{Px0x \leftarrow, Pxsyz \leftarrow Pxyz\}$$

Queremos saber si  $\mathbb{P}_+ \models Ps0s0w$ , es decir, cuánto es  $1 + 1$ .

El principio de refutación y la correctud de la resolución nos dice que basta agregar la meta  $\leftarrow Ps0s0w$  a  $\mathbb{P}_+$  y obtener la cláusula vacía.

1.  $Px0x \leftarrow$
2.  $Pxsyzs \leftarrow Pxyz$
3.  $\leftarrow Ps0s0w$
4.  $\leftarrow Ps00z \quad res(2, 3, [x, y, w := s0, 0, sz])$
5.  $\leftarrow \quad res(4, 1, [x, z := s0])$

De manera que  $Ps0s0w$  es consecuencia de  $\mathbb{P}_+$ . Más aún, la composición de los unificadores utilizados nos devuelve  $[w := ss0]$  que es la respuesta buscada.

**Ejemplo 2.2** Veamos ahora un programa para el producto de naturales:

$$\mathbb{P}_\times = \mathbb{P}_+ \cup \{M0x0 \leftarrow, Msxyz \leftarrow Mxyv, Pvyz\}$$

Nos preguntamos cuánto es  $1 \times 2$ , la consecuencia lógica asociada es  $\mathcal{P}_\times \models Ms0ss0w$

1.  $M0u0 \leftarrow$
2.  $Msxyz \leftarrow Mxyv, Pvyz$
3.  $Px_10x_1 \leftarrow$
4.  $Px_2sy_2sz_2 \leftarrow Px_2y_2z_2$
5.  $\leftarrow Ms0ss0w$
6.  $\leftarrow M0ss0v, Pvs0w \quad res(2, 5, [x, y, z := 0, ss0, w])$
7.  $\leftarrow P0ss0w \quad res(1, 6, [u, v := ss0, 0])$
8.  $\leftarrow P0s0z_2 \quad res(4, 7, [x_2, y_2, w := 0, s0, sz_2])$ ,
9.  $\leftarrow P00z_3 \quad res(4 \text{ renombrando con } [z_2 := z_3], 8, [x_3, y_3, z_2 := 0, 0, sz_3])$
10.  $\leftarrow \quad res(3, 9, [x_1, z_3 := 0, 0])$

Para obtener el valor de  $w$  componemos los valores necesarios de los unificadores utilizados, esto se conoce como *sustitución de respuesta*:

$$[w := sz_2][z_2 := sz_3][z_3 := 0], \text{ es decir }, [w := ss0]$$

Obsérvese que en la cláusula 6 existe un no determinismo, podemos resolver cualquiera de las dos literales. Si hubiéramos elegido resolver la segunda la derivación sería:

7.  $\leftarrow M0ss0v, Pvs0z_2 \quad res(4, 6, [x_2, y_2, w := v, s0, sz_2])$
8.  $\leftarrow M0ss0v, Pv0z_3 \quad res(4 \text{ renombrando con } [z_2 := z_3], 7, [x_2, y_2, z_2 := v, 0, sz_3])$
9.  $\leftarrow M0ss0v \quad res(3, 8, [x_1, z_3 := v, v])$
10.  $\leftarrow \quad res(1, 9, [u, v := ss0, 0])$

La sustitución de respuesta es:

$$[w := sz_2][z_2 := sz_3][z_3 := v][v := 0], \text{ es decir }, [w := ss0]$$

En este caso se obtuvo la misma respuesta en ambos casos, sin embargo, la programación lógica es sensible al orden de las cláusulas y al orden en que se eligen las literales para resolver. A continuación ejemplificamos este fenómeno.

**Ejemplo 2.3** Modificamos el programa para la suma de manera que la recursión sea en la primera variable.

$$\mathbb{P}'_+ = \{P0xx \leftarrow, Pxsyzs \leftarrow Pxyz\}$$

Semánticamente los resultados deben ser iguales, y lo son, pero operacionalmente existen grandes diferencias.

Veamos qué sucede ante la meta  $\leftarrow Pws0ss0$ . La respuesta buscada es  $w := s0$ .

1.  $Psx_2y_2sz_2 \leftarrow Px_2y_2z_2$
2.  $P0xx \leftarrow$
3.  $\leftarrow Pws0ss0$
4.  $\leftarrow Px_2s0s0 \quad res(1, 3, [w, y_2, z_2 := sx_2, s0, s0])$

En este punto hay dos elecciones para resolver 4, la regla 1 ó el hecho 2; como 1 se lista primero, se intenta con dicha regla:

5.  $\leftarrow Px_3s00 \quad res(1 \text{ renombrando con } [x_2 := x_3], 4, [x_2, y_2, z_2 := sx_3, s0, 0])$

En este punto no es posible resolver 5 y la búsqueda de  $\leftarrow$  ha fallado. Regresamos al último punto donde hubo una elección y elegimos de manera distinta, en este caso 4 con 2:

5.  $\leftarrow \quad res(2, 4, [x_2, x := 0, s0])$

La búsqueda tiene éxito y la sustitución de respuesta es  $[w := s0]$ .

Obsérvese en este ejemplo además que estamos usando un mismo programa para una tarea distinta a aquella por la cual se diseñó. El programa fue diseñado para sumar, pero también sirve para restar, el predicado  $Rxyz$  tal que  $z = x - y$  puede programarse como:

$$Rxyz \leftarrow Pyzx$$

Esto se conoce como uso no estándar de un programa lógico y es una característica exclusiva de este tipo de programas.

En el ejemplo anterior vimos que una elección inadecuada de una cláusula para resolver puede causar que falle la búsqueda de  $\leftarrow$ . Veamos ahora otro ejemplo que causa no terminación de la búsqueda.

**Ejemplo 2.4** Calculemos nuevamente  $1 \times 2$ , esta vez con el segundo programa para la suma

1.  $M0u0 \leftarrow$
2.  $Mszyz \leftarrow Mxyv, Pvyz$
3.  $P0x_1x_1 \leftarrow$
4.  $Psx_2y_2sz_2 \leftarrow Px_2y_2z_2$
5.  $\leftarrow Ms0ss0w$
6.  $\leftarrow M0ss0v, Pvs0w \quad res(2, 5, [x, y, z := 0, ss0, w])$

Para continuar, si elegimos para resolver ahora la segunda literal en 6, como no hay información para  $v, w$ , ambas cláusulas para la suma son aplicables. Si elegimos la cláusula 4 obtenemos:

7.  $\leftarrow M0ss0sx_2, Px_2ss0z_2 \quad res(6, 4, [v, w, y_2 := sx_2, sz_2, ss0])$

Como se observa obtuvimos la misma segunda literal con variables renombradas. Si nuevamente elegimos resolver la segunda literal de 7 con 4 obtenemos:

$$8. \quad \leftarrow M0ss0ssx_3, Px_3ss0z_3 \quad res(7, 4 \text{ renombrando con } [x_2, z_2 := x_3, z_3], [x_2, z_2, y_2 := sx_3, sz_3, ss0])$$

Si seguimos usando la misma elección la búsqueda no terminará jamás aún cuando  $\square$  sí puede obtenerse. Más aún, al cambiar la elección a la primera literal de la meta, la respuesta será *No* pues la meta no se puede resolver. En conclusión, un cambio en el orden de las cláusulas puede causar no terminación o bien una respuesta negativa a pesar de que una respuesta afirmativa existe.

### 3. Ejercicios

1. Defina la función de exponenciación de naturales, en lógica de predicados y como programa lógico.
2. Defina la función diferencia de naturales, como programa lógico.
3. Defina la función diferencia de naturales como programa lógico, utilizando la función suma exclusivamente.
4. Dé una especificación no recursiva en lógica de predicados de la relación de orden  $<$ , utilizando la función suma. Transforme dicha especificación a un programa lógico.
5. Considere la siguiente especificación de estructuras de datos para números positivos y números naturales en representación por paridad.
  - El uno es un número positivo.
  - El doble de un número positivo es un número positivo.
  - El sucesor del doble de un número positivo es un número positivo.
  - El cero es un número natural (por paridad).
  - Un número positivo es un número natural (por paridad).
  - a) Realice la especificación formal en lógica de predicados usando el siguiente glosario:
    - $Pos(x) =_{def} x$  es positivo
    - $PNat(x) =_{def} x$  es natural (por paridad)
    - $d(x) =_{def}$  el doble de  $x$  (i.e  $d(x) = 2x$ ,  $d$  es símbolo de función)
    - $o(x) =_{def}$  el sucesor del doble de  $x$  (i.e  $o(x) = 2x + 1$ ,  $o$  es símbolo de función)
    - Las constantes 0 y 1.
  - b) Especifique las operaciones suma y producto para naturales con paridad. En cada caso dé tres versiones en lógica de predicados, en forma cláusular y como programa lógico.
  - c) Especifique las operaciones de conversión entre naturales usuales  $Nat$  y naturales por paridad  $PNat$ . En cada caso dé tres versiones en lógica de predicados, en forma cláusular y como programa lógico.
6. ¿Cuáles de los programas lógicos de los ejercicios anteriores *no* son programas definidos ?