

# GEOMETRÍA PLANA

Una introducción moderna con programación

Alejandro Sánchez Yalí

PRIMERA EDICIÓN

# Índice general

<b>1. Preliminares</b>	<b>1</b>
1.1. Sistemas formales . . . . .	1
1.1.1. El Acertijo MU: Un enfoque computacional . . . . .	2
<b>Apéndice: Material Complementario</b>	<b>7</b>
.1. Demostraciones Adicionales . . . . .	7
.2. Tablas de Datos . . . . .	7

# Índice de figuras

1.1. Árbol de cadenas bien formadas (teoremas) en el sistema formal $MIU$ . . .	3
---	---

# Índice de cuadros

# Capítulo 1

## Preliminares

La Geometría es, posiblemente, la rama más antigua de las matemáticas. Sus aplicaciones fueron estudiadas hace miles de años por egipcios, babilonios y chinos, mientras que su teoría fue desarrollada por los griegos. Lo que distingue el estudio moderno de la geometría del antiguo es el *método axiomático*, piedra angular sobre la cual descansa toda la matemática pura. En este capítulo exploraremos las bases de los sistemas formales y cómo estos conceptos teóricos se traducen en algoritmos ejecutables.

### 1.1 Sistemas formales

El núcleo metodológico de las matemáticas reside en el **método axiomático**. Este enfoque se fundamenta en la postulación de un conjunto inicial de enunciados denominados **axiomas**, los cuales deben ser mutuamente consistentes e idealmente independientes entre sí. Este cuerpo de proposiciones conforma lo que denominamos un **sistema axiomático** o **sistema formal**, donde el punto de partida de toda cadena deductiva son los axiomas: proposiciones que se asumen como verdaderas sin necesidad de demostración previa.

¿Por qué necesitamos axiomas? La respuesta está en un problema lógico fundamental: no es posible justificar una afirmación partiendo de la nada. Si una proposición se sostiene en otra, entonces surge la pregunta de cómo se justifica esta última. A su vez, necesitaría apoyarse en una tercera, y así sucesivamente. Si seguimos este razonamiento, nos encontramos con una regresión infinita de justificaciones, como una cadena interminable de dominós. Para evitar este problema, es necesario aceptar ciertas proposiciones como verdaderas sin necesidad de demostración. Estas proposiciones son los axiomas.

El método axiomático puede resumirse en los siguientes principios:

1. Todo sistema axiomático debe contener un conjunto de **términos primitivos**, deliberadamente elegidos como indefinidos y sujetos a la interpretación del lector.
2. Todos los demás términos técnicos del sistema se definen a partir de los términos primitivos. Estos son las **definiciones** del sistema.
3. El sistema contiene un conjunto de enunciados sobre términos primitivos y definiciones que se eligen sin demostrar. Estos son los **axiomas** del sistema.

4. Todos los demás enunciados del sistema deben ser consecuencias lógicas de los axiomas. Estos enunciados derivados se denominan **teoremas**.

Desde una perspectiva computacional, un sistema formal es un mecanismo de manipulación de signos puramente sintáctico. Está constituido por un alfabeto de símbolos que se **concatenan** para formar secuencias (o cadenas), las cuales son manipuladas siguiendo reglas estrictas para producir nuevas secuencias. Esta capacidad de manipulación simbólica permite que el sistema funcione como una representación estructurada, permitiéndonos explorar relaciones lógicas y deducciones mediante razonamientos formales.

En disciplinas como la informática, la teoría de la información y la estadística, estos sistemas actúan como gramáticas formales diseñadas para la modelización. Al proceso de construir estos sistemas se le llama **formalización**: un acto de abstracción mediante el cual estructuramos conceptos y relaciones en un lenguaje formal riguroso.

Para implementar computacionalmente un sistema formal, se requiere definir cuatro elementos:

1. Un **alfabeto** de símbolos primitivos.
2. Un conjunto de reglas gramaticales para definir las **fórmulas bien formadas** (cadenas válidas).
3. Un conjunto de fórmulas bien formadas iniciales llamadas **axiomas**.
4. Un conjunto finito de **reglas de inferencia** para derivar nuevos teoremas.

### 1.1.1. El Acertijo MU: Un enfoque computacional

En esta sección estudiaremos el acertijo MU, el cual representa un sistema formal minimalista. Este problema fue planteado por Douglas Hofstadter en 1979 en su obra seminal *Gödel, Escher, Bach: Un Eterno y Grácil Bucle*<sup>1</sup>.

El objetivo es producir la cadena MU (de ahí su nombre) dentro de un sistema formal conocido como el **sistema MIU**. El nombre deriva de su alfabeto, compuesto exclusivamente por tres símbolos: M, I, U. Esto significa que las cadenas del sistema estarán formadas únicamente por combinaciones de estas tres letras.

#### Definición del sistema MIU

El sistema parte de una cadena inicial o axioma: la cadena MI. A partir de ella, se pueden generar nuevas cadenas aplicando reglas de transformación. Si definimos  $x$  e  $y$  como variables que representan cualquier cadena de símbolos del sistema, las reglas son:

- **Regla 1 (Adición de U).** Si una cadena termina en I, se puede agregar una U al final.

$$xI \implies xIU$$

*Ejemplo:* Si tenemos MII, podemos derivar MIIU.

---

<sup>1</sup>Una lectura recomendada para entender la recursividad y la inteligencia artificial.

- **Regla 2 (Duplicación).** Si una cadena comienza con M, se puede duplicar todo lo que sigue.

$$Mx \implies Mxx$$

*Ejemplo:* De MIU se deriva MIUIU.

- **Regla 3 (Reducción de III).** Si aparece la secuencia III, se puede sustituirse por U.

$$xIIIy \implies xUy$$

*Ejemplo:* De UMIIIMU se deriva UMUMU. Nótese que las tres I deben ser consecutivas.

- **Regla 4 (Eliminación de UU).** Si aparece la secuencia UU, está permitida su eliminación.

$$xUUy \implies xy$$

*Ejemplo:* De MUUUIII se deriva MUIII.

## Exploración del espacio de búsqueda

Generar teoremas en un sistema formal es equivalente a realizar una *búsqueda en un espacio de estados*, donde cada estado es una cadena válida y las transiciones son las reglas de inferencia. Esta perspectiva computacional justifica el uso de algoritmos de búsqueda para explorar el sistema.

Para intentar resolver el acertijo, procedemos a generar cadenas manualmente partiendo del axioma MI:

1. Aplicando la Regla 2 a MI, obtenemos MII.
2. Aplicando la Regla 1 a MI, obtenemos MIU.

Este proceso genera una estructura de árbol (Figura 1.1). Sin embargo, a diferencia de los árboles binarios simples, aquí el factor de ramificación es variable. Por ejemplo, la cadena MIIIIIIIIU permite múltiples aplicaciones de la Regla 3 en diferentes posiciones, generando diversos hijos.

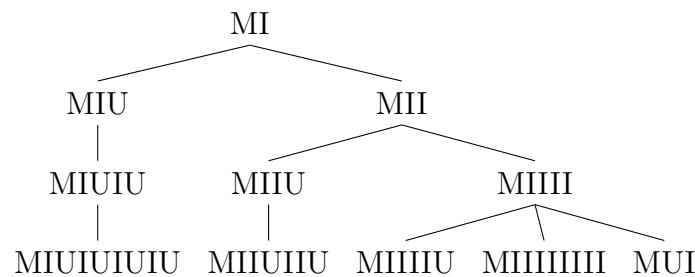


Figura 1.1: Árbol de cadenas bien formadas (teoremas) en el sistema formal *MIU*.

## Implementación algorítmica

Dado que el árbol de derivación crece exponencialmente, la exploración manual se vuelve inviable rápidamente. Aquí es donde la programación se convierte en una herramienta indispensable para las matemáticas.

El algoritmo para generar todas las derivaciones posibles de una cadena se puede expresar de manera abstracta mediante el siguiente pseudocódigo:

---

### Algoritmo 1: Derivaciones en el sistema MIU

---

**Entrada:** Una cadena  $s$  del sistema MIU  
**Salida:** Conjunto de todas las cadenas derivables en un paso

$D \leftarrow \emptyset$ ;  
 // Regla 1:  $xI \Rightarrow xIU$   
**si**  $s$  termina en  $I$  **entonces**  
    $D \leftarrow D \cup \{s + U\}$ ;  
 // Regla 2:  $Mx \Rightarrow Mxx$   
**si**  $s$  comienza con  $M$  **entonces**  
   Sea  $x$  la subcadena después de  $M$ ;  
    $D \leftarrow D \cup \{M + x + x\}$ ;  
 // Regla 3:  $xIIIy \Rightarrow xUy$   
**para cada** posición  $i$  donde aparece  $III$  en  $s$  **hacer**  
    $D \leftarrow D \cup \{s[0:i] + U + s[i+3:]\}$ ;  
 // Regla 4:  $xUUy \Rightarrow xy$   
**para cada** posición  $i$  donde aparece  $UU$  en  $s$  **hacer**  
    $D \leftarrow D \cup \{s[0:i] + s[i+2:]\}$ ;  
**retornar**  $D$ ;

---

A continuación, traducimos este pseudocódigo a Python:

### Código 1.1: Implementación de las reglas del sistema MIU en Python

```
1 def aplicar_reglas(cadena):
2     """
3     Recibe una cadena del sistema MIU y devuelve un conjunto
4     con todas las cadenas que se pueden derivar de ella en un paso.
5     """
6     derivaciones = set()
7
8     # Regla 1: xI -> xIU
9     if cadena.endswith('I'):
10         derivaciones.add(cadena + 'U')
11
12     # Regla 2: Mx -> Mxx
13     if cadena.startswith('M'):
14         x = cadena[1:]
15         derivaciones.add('M' + x + x)
16
17     # Regla 3: III -> U (Buscamos en todas las posiciones posibles)
```

```

18     for i in range(len(cadena) - 2):
19         if cadena[i:i+3] == 'III':
20             # Reconstruimos la cadena reemplazando III por U
21             nueva = cadena[:i] + 'U' + cadena[i+3:]
22             derivaciones.add(nueva)
23
24     # Regla 4: UU -> eliminar (Buscamos en todas las posiciones
25     # posibles)
26     for i in range(len(cadena) - 1):
27         if cadena[i:i+2] == 'UU':
28             # Reconstruimos la cadena eliminando UU
29             nueva = cadena[:i] + cadena[i+2:]
30             derivaciones.add(nueva)
31
32     return derivaciones
33
34 # Prueba del axioma inicial
35 axioma = "MI"
36 hijos = aplicar_reglas(axioma)
37 print(f"Del axioma '{axioma}' se derivan: {hijos}")

```

Este código nos permite automatizar la expansión del árbol y explorar niveles más profundos del espacio de búsqueda.

## La imposibilidad de MU

Después de explorar el árbol de derivaciones, surge una pregunta natural: ¿es posible alcanzar MU? La respuesta es **no**, y la demostración es un ejemplo elegante de razonamiento matemático.

La clave está en observar un **invariante**: una propiedad que se preserva bajo todas las reglas del sistema. Contemos las letras *I* en cada cadena y clasifiquémoslas según el **residuo** al dividir entre 3. Por ejemplo: una letra *I* deja residuo 1; dos letras dejan residuo 2; tres letras dejan residuo 0; cuatro letras dejan residuo 1 (porque  $4 = 3 + 1$ ), y así sucesivamente.

Analicemos cómo cada regla afecta este residuo:

- **Estado inicial:** La cadena MI tiene una sola letra *I*, por lo que su residuo es 1.
- **Regla 1** (añadir U): No altera el número de letras *I*, así que el residuo permanece igual.
- **Regla 2** (duplicar): Duplica las *I*. Si el residuo era 1, pasa a ser 2. Si era 2, pasa a ser 1 (porque  $2 \times 2 = 4$ , y 4 deja residuo 1). Nótese que el residuo 0 pasaría a 0.
- **Regla 3** ( $III \rightarrow U$ ): Elimina exactamente tres letras *I*, lo cual no cambia el residuo (restar 3 no afecta el residuo módulo 3).
- **Regla 4** (eliminar UU): No involucra letras *I*, por lo que el residuo no cambia.

Partiendo de residuo 1, solo podemos alcanzar residuos 1 o 2, pero **nunca 0**. La cadena MU no tiene letras I (residuo 0), lo cual es imposible de alcanzar desde MI.

Este resultado ilustra un principio fundamental: a veces, la exploración computacional no basta. Se requiere un argumento matemático que trascienda la búsqueda exhaustiva. El acertijo MU nos enseña que trabajar *dentro* de un sistema formal tiene límites, y que para entender verdaderamente sus propiedades debemos *salir* de él y razonar desde una perspectiva externa.

*Ejercicio 1.1.* Implementa en Python una función `es_derivable(cadena)` que determine si una cadena dada es alcanzable desde MI en el sistema MIU. La función debe:

1. Verificar que la cadena comience con M y solo contenga los símbolos M, I, U.
2. Contar el número de letras I en la cadena.
3. Calcular el residuo de ese número al dividir entre 3.
4. Retornar `True` si el residuo es 1 o 2, y `False` si es 0.

Prueba tu función con las cadenas MU, MIU, MIUIU y MUII.

*Ejercicio 1.2.* Considera un sistema axiomático minimalista con dos tipos de objetos: **Fe's** y **Fo's**. Los axiomas del sistema son:

1. Existen exactamente tres Fe's.
2. Para cada par de Fe's distintos, existe exactamente un Fo al cual ambos pertenecen.
3. No todos los Fe's pertenecen al mismo Fo.
4. Dos Fo's distintos tienen al menos un Fe en común.

A partir de estos axiomas, demuestra las siguientes proposiciones:

- (a) Cada Fo tiene exactamente dos Fe's que le pertenecen.
- (b) Un Fo está completamente caracterizado por los Fe's que contiene. Es decir, si dos Fo's son distintos, entonces no contienen los mismos Fe's.
- (c) Dos Fo's distintos tienen exactamente un Fe en común.
- (d) Existen exactamente tres Fo's distintos.

# Apéndice: Material Complementario

## **.1 Demostraciones Adicionales**

Aquí puedes incluir demostraciones más detalladas o extensas que no encajan en el cuerpo principal del libro.

## **.2 Tablas de Datos**

Aquí puedes incluir tablas de datos extensas o información de referencia.