Lógica formal, verificación automática de teoremas y asistentes de demostración

Carlos Paesa Lía

Trabajo de fin de grado de Matemáticas
Universidad de Zaragoza

Director del trabajo: Miguel Ángel Marco Buzunámiz
3 de septiembre de 2023

Preface

Mathematics as a whole relies heavily on proofs and their validity, so the moment had to come when mathematicians would need to focus on the way logic and proofs work. In this work, we aim to give an explanation of how logical reasoning can be formalised, how it relates to type theory and how proof assistants can be used for automatic theorem checking. In the first chapter we will understand what a formal system is and work with the most basic example: propositional logic. The second chapter will extend the notions seen in the first one in order to proof Gödel's Completeness Theorem and other interesting properties for first order logic, ending with an introduction to higher-order systems. The last chapter introduces briefly the main concepts in type theory, in order to explain the Curry-Howard correspondence between models of computation (type theory) and the proof systems we talked about in the first two chapters.

First we define the concept of a *formal system* and the concepts related to them: formulae, axioms, hypothesis, inference rules, deduction and proofs. As the main example, we define *propositional logic* with implication ($\rightarrow$) and negation ($\neg$) and explain how proofs are built, using a "tower" notation. Then we take a look at other *connectors* such as conjunction ($\wedge$), disyunction ($\vee$) and double implication ($\leftrightarrow$), together with their respective introduction and elimination rules of inference. Before we move from syntax to semantics, we take a look into the system we refer to as "classical reasoning", with the *law of excluded middle* and *proof by contradiction*. Then we define semantic concepts that arise such as truth values, tautologies, *soundness* and *completeness*, in order to proof (for propositional logic) a very natural intuition: we expect statements we can proof to always be true and, the other way around, true statements to always be provable.

Since these tools are not enough to describe all logical conclusions, we extend the definition to *first-order systems*, where we can talk about properties of the elements: variables, predicates, relationships and functions. We can also define equality and quantify over variables. We do so with the existential ($\exists$) and universal quantifiers ($\forall$), both of which have their own inference rules to work with. With these new concepts, we construct *first-order logic* and make sense of the new notion of truth in this system. In order to redefine tautologies, we talk about interpretations and models that extend the notion of truth seen in the first chapter. Next we give a proof of Gödel's Completeness Theorem and the soundness of first-order logic, discussing the reason why this does not contradict Gödel's Incompleteness Theorem. We end the chapter introducing the ideas behind higher-order systems and the abstraction capacity they enable.

The last section consists of a less rigorous explanation of type theory. Using a few examples, we show the ideas it builds upon and then introduce the basic types normally used. But, ¿what does type theory have to do with formal logic? As the Curry-Howard Correspondence shows, these two distinct conceptual frameworks are indeed the same. This means essencially that *"a proof is a program, and the formula it proves is the type for the program"*. As mathematics has grown in complexity and depth, so has the demand for automatic tools that can help navigate the mathematical reasoning behind many proofs. Proof assistants (most of which are type checkers in disguise) are being used to verify and store a significant part of the current mathematical knowledge in community-made libraries. At the end of this work, we talk about *Lean*, explaining step by step how to write proofs and building the natural numbers and their properties as an example in the Annex.

Note that the reader does not need previous knowledge about formal logic or type theory, besides basic ideas about proofs and logical thinking.

The main ideas in the first two chapters of this work are taken from [2], while most proofs and

definitions come from [1] and [6]. The last chapter is inspired by [7] with ideas and concepts from [5].

I would like to thank my tutor Miguel Ángel Marco for the helpful feedback and the proposal of this interesting and different topic, since during my studies there was no opportunity to learn about formal logic and/or type theory.

Índice general

Capítulo 1

El lenguaje de la lógica

En el día a día utilizamos la lógica para relacionar enunciados de forma general, sin tener en cuenta su significado concreto. Podemos describir este comportamiento de forma abstracta mediante un sistema axiomático.

Los sistemas axiomaticos describen sistemas deductivos. A partir de unos axiomas y *reglas de inferencia*, se construyen todos los elementos del sistema.

Definición 1.1. Un **sistema formal S** es una estructura abstracta $S = (\mathcal{A}, \Omega, I, Z)$ formada por:

1. $\mathcal{A}$, un conjunto finito de símbolos (alfabeto).
2. Ω, un conjunto de conectores lógicos. Una *fórmula* es un símbolo del alfabeto o la combinación recursiva de fórmulas mediante los conectores lógicos.
3. I, un conjunto de axiomas formado por fórmulas.
4. Z, un conjunto de reglas de inferencia, que permiten obtener fórmulas a partir de otras (*derivación*).

1.1. La lógica proposicional

El sistema más básico es el de la lógica proposicional, que puede definirse de varias maneras, entendiendo que existen correspondencias (isomorfismos) entre las definiciones que hacen que sean equivalentes.

Una posible construcción es:

1. $\mathcal{A} = \{p, q, r, s, t, \dots\}$
2. $\Omega = \{\bot, \top, \neg, \to\}$, siendo la falsedad lógica ($\bot$), la verdad lógica ($\top$), la negación ($\neg$) y la implicación ($\to$).
3. Los axiomas siguientes respecto a la negación e implicación, para fórmulas A, B, C cualesquiera:

   $$(\to 1) \qquad A \to (B \to A)$$
   $$(\to 2) \qquad (A \to (B \to C)) \to ((A \to B) \to (A \to C))$$
   $$(\to 3) \qquad (\neg B \to \neg A) \to ((\neg B \to A) \to B)$$

   Como consecuencia de estos axiomas, los operadores se comportan como la definición usual de negación e implicación.
4. La regla de inferencia *Modus Ponens*. Si A, B son fórmulas cualesquiera, a partir de $A, A \to B$ inferimos B:

   $$\frac{A \to B \qquad A}{B} \to \text{E}$$

Representamos las hipótesis por separado encima de una línea horizontal y debajo la conclusión. Marcamos la regla a la derecha con "$\to E$" porque en esencia se trata de la eliminación de la implicación en una fórmula. Esta notación permite generalizar otras reglas de inferencia como reglas de eliminación o de inclusión de otros operadores lógicos. Hablaremos más de ello en el Apartado 1.2

Llamaremos a partir de ahora L_P al sistema de la lógica proposicional definido así.

Definición 1.2. Una **prueba P** dentro de un sistema formal es una secuencia finita de *fórmulas* $F_1, \ldots, F_n$ de manera que cada una es un axioma ($F_i \in I$) o una derivación a partir de una regla de inferencia de Z y un conjunto de fórmulas anteriores. Decimos que P es una prueba de F_n.

Las representaremos recursivamente con las fórmulas separadas sobre una línea horizaontal y la derivación a partir de reglas de inferencia debajo.

Definición 1.3. Una fórmula F se dice **demostrable** si existe P prueba de F. Lo denotamos $\vdash_S F$, omitiendo el sistema formal S cuando esté claro. Si una fórmula no es demostrable, lo denotamos $\nvdash_S F$

Ejemplo 1.4. A partir de axiomas y reglas de inferencia, podemos construir pruebas dentro del sistema, por ejemplo, de que $A \to A$ para A fórmula cualquiera de L_P.

(F_1) Axioma 2: $(A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A))$

(F_2) Axioma 1: $A \to ((A \to A) \to A)$

(F_3) Modus Ponens, F_1, F_2: $(A \to (A \to A)) \to (A \to A)$

(F_4) Axioma 1: $A \to (A \to A)$

(F_5) Modus Ponens, F_3, F_4: $A \to A$

Nota 1.5. A continuación, las pruebas las construiremos gráficamente a partir de su secuencia. Los axiomas los introduciremos con una línea horizontal encima en la que marcamos el número del axioma a la derecha. *Modus Ponens* lo aplicaremos poniendo una línea horizontal bajo las dos fórmulas escribiendo abajo el resultado y "$\to E$" a la derecha. En general ahí escribiremos la regla de inferencia utilizada.

Con esta notación, la prueba queda de la siguiente manera:

$$\dfrac{\dfrac{\dfrac{}{(A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A))}\,(\to 2) \qquad \dfrac{}{A \to ((A \to A) \to A)}\,(\to 1)}{(A \to (A \to A)) \to (A \to A)}\,{\to}\mathrm{E} \qquad \dfrac{}{A \to (A \to A)}\,(\to 1)}{A \to A}\,{\to}\mathrm{E}$$

En ocasiones trabajamos con un conjunto de fórmulas que damos por válidas y a partir de ellas obtenemos resultados. Eso motiva la siguiente definición.

Definición 1.6. Dado un sistema formal S, una fórmula A y Γ un conjunto de fórmulas de S (hipótesis), decimos que A **se deriva** de Γ en S si existe una prueba P de A utilizando Γ también como axiomas. Lo denotamos $\Gamma \vdash_S A$. Si por el contrario A no se deriva de Γ, lo denotamos $\Gamma \nvdash_S A$.

Observación 1.7. Si consideramos sistemas formales $S_1 = (\mathcal{A}, \Omega, I, Z)$ y $S_2 = (\mathcal{A}, \Omega, \emptyset, Z)$, decir que $\vdash_{S_1} A$ es lo mismo que decir que $I \vdash_{S_2} A$. Por tanto, axiomas e hipótesis son en realidad intercambiables.

Teorema 1.8 (Teorema de la deducción (TD))**.** *Sea $S = L_P$, A, B fórmulas y Γ un conjunto de fórmulas de L_P tales que $\Gamma \cup \{A\} \vdash B$. Entonces $\Gamma \vdash A \to B$. Decimos que A es una hipótesis eliminada.*

Demostración. Por inducción sobre la longitud n de la prueba de B a partir de $\Gamma \cup \{A\}$. La hipótesis de inducción es que para Γ, A, B tales que existe una prueba de B a partir de $\Gamma \cup \{A\}$ de logitud menor o igual que n, se tiene que $\Gamma \vdash A \to B$.

Si $n = 1$, entonces B está en $\Gamma \cup \{A\}$. Tenemos dos casos:

1. $B = A$ y $\Gamma \vdash A \to A$ por el Ejemplo 1.4.

2. $B \in \Gamma$ y por tanto $\Gamma \vdash B$. Por el Axioma $(\to 1)$ se tiene que $\Gamma \vdash B \to (A \to B)$ y aplicando Modus Ponens, se llega a que $\Gamma \vdash A \to B$.

Asumiendo la hipótesis de inducción, sea B fórmula con una prueba a partir de $\Gamma \cup \{A\}$ de logitud $n+1$. Tenemos dos posibilidades:

1. B está en $\Gamma \cup \{A\}$ y procedemos como para $n = 1$.

2. B se infiere mediante Modus Ponens. Entonces existe una fórmula C tal que $\Gamma \cup \{A\}$ prueba C y $C \to B$, a partir de las cuales se aplica Modus Ponens. Ambas pruebas tendrán a lo sumo longitud n, luego por hipótesis de inducción $\Gamma \vdash A \to C$ y $\Gamma \vdash A \to (C \to B)$. Por el Axioma $(\to 2)$ se tiene que $\Gamma \vdash (A \to (C \to B)) \to ((A \to C) \to (A \to B))$. Basta aplicar dos veces Modus Ponens para obtener que $\Gamma \vdash A \to B$.

Por tanto se cumple también para $n+1$ y por inducción queda demostrado el resultado. □

Nota 1.9. Este teorema nos es útil en la construcción de pruebas. Siguiendo la notación introducida en la Nota 1.5, cuando lo utilicemos, pondremos una línea horizontal sobre la hipótesis eliminada A y a la derecha escribiremos un número identificativo. Al final de la prueba colocaremos una línea horizontal bajo B, a la derecha escribiremos TD y el mismo número identificativo y debajo pondremos $A \to B$.

Ejemplo 1.10. Supongamos que $A \vdash B$, con la notación descrita, la aplicación del Teorema 1.8 quedaría:

$$\dfrac{\dfrac{}{A}\,1}{\begin{array}{c}\vdots\\ B\end{array}}\,1\ \text{TD}$$

Proposición 1.11 (Propiedades)**.** *Sean A, B, C fórmulas de L_P, entonces:*

1. $A \to B, B \to C \vdash A \to C$
2. $A \to (B \to C), B \vdash A \to C$
3. $\vdash \neg\neg A \to A$
4. $\vdash A \to \neg\neg A$
5. $\vdash \neg A \to (A \to B)$
6. $\vdash (\neg B \to \neg A) \to (A \to B)$
7. $\vdash (A \to B) \to (\neg B \to \neg A)$
8. $\vdash A \to (\neg B \to \neg(B \to A))$
9. $\vdash (A \to B) \to ((\neg A \to B) \to B)$

Demostración.

1.

$$\dfrac{\dfrac{\dfrac{}{A}\,1 \qquad A \to B}{B}\,{\to}\text{E} \qquad B \to C}{\dfrac{C}{A \to C}\,1\ \text{TD}}\,{\to}\text{E}$$

2.

$$\dfrac{\dfrac{\dfrac{}{A}\,1 \qquad A \to (B \to C)}{B \to C}\,{\to}\text{E} \qquad B}{\dfrac{C}{A \to C}\,1\ \text{TD}}\,{\to}\text{E}$$

3.

$$\dfrac{\dfrac{\dfrac{}{(\neg A \to \neg\neg A) \to ((\neg A \to \neg A) \to A)}\,(\to 3) \qquad \dfrac{}{\neg A \to \neg A}\,\text{Ej. 1.4}}{(\neg A \to \neg\neg A) \to A}\,\text{Prop. 1.11-2} \qquad \dfrac{}{\neg\neg A \to (\neg A \to \neg\neg A)}\,(\to 1)}{\neg\neg A \to A}\,\text{Prop. 1.11-1}$$

4.

$$\dfrac{\dfrac{\dfrac{}{(\neg\neg\neg A \to \neg A) \to ((\neg\neg\neg A \to A) \to \neg\neg A)}\,(\to 3) \qquad \dfrac{}{\neg\neg\neg A \to \neg A}\,\text{Prop. 1.11-3}}{(\neg\neg\neg A \to A) \to \neg\neg A}\,{\to}\text{E} \qquad \dfrac{}{A \to (\neg\neg\neg A \to A)}\,(\to 1)}{A \to \neg\neg A}\,\text{Prop. 1.11-1}$$

5.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A}\,1 \qquad \dfrac{}{A \to (\neg B \to A)}\,(\to 1)}{\neg B \to A}\,{\to}\text{E} \qquad \dfrac{\dfrac{\dfrac{}{\neg A}\,2 \qquad \dfrac{}{\neg A \to (\neg B \to \neg A)}\,(\to 1)}{\neg B \to \neg A}\,{\to}\text{E} \qquad \dfrac{}{(\neg B \to \neg A) \to ((\neg B \to A) \to B)}\,(\to 3)}{(\neg B \to A) \to B}\,{\to}\text{E}}{B}\,{\to}\text{E}}{A \to B}\,1\,\text{TD}}{\neg A \to (A \to B)}\,2\,\text{TD}$$

6.

$$\dfrac{\dfrac{\dfrac{}{\neg B \to \neg A}\,1 \qquad \dfrac{}{(\neg B \to \neg A) \to ((\neg B \to A) \to B)}\,(\to 3)}{(\neg B \to A) \to B}\,{\to}\text{E} \qquad \dfrac{}{A \to (\neg B \to A)}\,(\to 1)}{A \to B}\,\text{Prop. 1.11-1}}{(\neg B \to \neg A) \to (A \to B)}\,1\,\text{TD}$$

7.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \to B}\,1 \qquad \dfrac{}{\neg\neg A \to A}\,\text{Prop. 1.11-3}}{\neg\neg A \to B}\,\text{Prop. 1.11-1} \qquad \dfrac{}{B \to \neg\neg B)}\,\text{Prop. 1.11-4}}{\neg\neg A \to \neg\neg B}\,\text{Prop. 1.11-1} \qquad \dfrac{}{(\neg\neg A \to \neg\neg B) \to (\neg B \to \neg A)}\,\text{Prop. 1.11-6}}{\neg B \to \neg A}\,{\to}\text{E}}{(A \to B) \to (\neg B \to \neg A)}\,1\,\text{TD}$$

8.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A}\,2 \qquad \dfrac{}{A \to B}\,1}{B}\,{\to}\text{E}}{(A \to B) \to B}\,1\,\text{TD}}{A \to ((A \to B) \to B)}\,2\,\text{TD} \qquad \dfrac{}{((A \to B) \to B) \to (\neg B \to \neg(A \to B))}\,\text{Prop. 1.11-7}}{A \to (\neg B \to \neg(A \to B))}\,\text{Prop. 1.11-1}$$

9.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg A \to B} \qquad \dfrac{}{(\neg A \to B) \to (\neg B \to \neg\neg A)}\,\text{Prop. 1.11-7}}{\neg B \to \neg\neg A}\,{\to}\text{E} \qquad \dfrac{}{(\neg B \to \neg\neg A) \to ((\neg B \to \neg A) \to B)}\,(\to 3)}{(\neg B \to \neg A) \to B}\,{\to}\text{E}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \to B}\,2 \qquad \dfrac{}{(A \to B) \to (\neg B \to \neg A)}\,\text{Prop. 1.11-7}}{\neg B \to \neg A}\,{\to}\text{E} \qquad \begin{array}{c}\dfrac{}{\neg A \to B}\,1\\ \vdots\\ (\neg B \to \neg A) \to B\end{array}}{B}\,{\to}\text{E}}{(\neg A \to B) \to B}\,1\,\text{TD}}{(A \to B) \to ((\neg A \to B) \to B)}\,2\,\text{TD}$$

□

1.2. Deducción natural y razonamiento clásico

En el día a día los razonamientos lógicos no suelen partir de axiomas, sino utilizar reglas de inferencia que permiten introducir y eliminar los conectores lógicos, que funcionan de forma más natural.

Definición 1.12. Decimos que un sistema formal $S = (\mathcal{A}, \Omega, I, Z)$ utiliza la **deducción natural** si $I = \emptyset$.

Así, otra posible definición para la lógica proposicional sería con $\mathcal{A} = \{p, q, r, s, t, \dots\}$, $\Omega = \{\bot, \top, \neg, \vee, \wedge, \to, \leftrightarrow\}$, $I = \emptyset$ y Z el conjunto de reglas de introducción y eliminación de los operadores de Ω. A este sistema lo denotaremos como L_I.

En esencia las reglas pueden ser de introducción o de elminación. Dado un conector lógico y a partir de un conjunto de fórmulas y/o pruebas, las primeras producen una fórmula con una aparición más de este y las segundas producen una fórmula con una aparición menos.

Nota 1.13. Para las reglas de inferencia, utilizaremos la misma notación descrita en las Notas 1.5 y 1.9 con números identificativos y en lugar de "$\to E$" o "TD" escribiremos la abreviatura de la regla concreta. En general usaremos como abreviatura el conector lógico junto a una I mayúscula para las reglas de introducción o una E mayúscula para las reglas de eliminación.

Los operadores nuevos representan la conjunción lógica ($\wedge$), la disyunción lógica ($\vee$) y la doble implicación ($\leftrightarrow$), que son equivalentes a fórmulas construidas utilizando solamente $\neg$ y $\to$, puesto que $\{\neg, \to\}$ es un conjunto de conectores lógicos funcionalmente completo (permite expresar todas las posibles tablas de verdad). Añadimos también como ejemplo el operador disyunción exclusiva ($\oplus$), aunque no lo utilizaremos:

- $A \wedge B \equiv \neg(A \to \neg B)$
- $A \vee B \equiv \neg A \to B$
- $A \leftrightarrow B \equiv (A \to \neg B) \to \neg(\neg A \to B)$
- $A \oplus B \equiv \neg((A \to \neg B) \to \neg(\neg A \to B))$

Como no utilizamos ningún axioma, para cada operador binario tenemos al menos dos reglas, una para introducir el operador y otra para eliminarlo de la fórmula:

- Implicación:

$$\dfrac{\begin{array}{c}\overline{A}^{\,1}\\ \vdots\\ B\end{array}}{A \to B}\,1\,{\to}\mathrm{I} \qquad \dfrac{A \to B \qquad A}{B}\,{\to}\mathrm{E}$$

  En realidad estas dos reglas ya las hemos visto: la introducción es el Teorema de la Deducción y la eliminación es *Modus Ponens*. La primera la interpretamos como que si tenemos una prueba P de B a partir de A, entonces podemos producir una prueba de $A \to B$. La segunda como que si tenemos P_1 prueba de $A \to B$ y P_2 prueba de A, podemos combinarlas para obtener una prueba de B.

- Negación:

$$\dfrac{\begin{array}{c}\overline{A}^{\,1}\\ \vdots\\ \bot\end{array}}{\neg A}\,1\,\neg\mathrm{I} \qquad \dfrac{\neg A \qquad A}{\bot}\,\neg\mathrm{E}$$

  En la introducción, si asumimos temporalmente A y llegamos a una contradicción, podemos concluir $\neg A$. En la eliminación, a partir de una fórmula y su negación, obtenemos una contradicción: $\bot$.

- Conjunción:

$$\frac{A \qquad B}{A \wedge B}\,\wedge\mathrm{I} \qquad \frac{A \wedge B}{A}\,\wedge\mathrm{E}_i \qquad \frac{A \wedge B}{B}\,\wedge\mathrm{E}_d$$

La forma de leer la regla de introduccion de la conjunción es que si tenemos P_1 prueba de A y P_2 prueba de B a partir de un conjunto de hipótesis, podemos combinarlas usando la regla para obtener una prueba de $A \wedge B$. De igual manera, una prueba de $A \wedge B$ es una prueba de A o de B y lo marcamos con el subíndice para diferenciar cual.

- Disyunción:

$$\frac{A}{A \vee B}\,\vee\mathrm{I}_i \qquad \frac{B}{A \vee B}\,\vee\mathrm{I}_d \qquad \frac{A \vee B \qquad \begin{array}{c}\overline{A}^{\,1}\\ \vdots \\ C\end{array} \qquad \begin{array}{c}\overline{B}^{\,1}\\ \vdots \\ C\end{array}}{C}\,1\vee\mathrm{E}$$

Las reglas de introducción expresan que a partir de una prueba P_1 de A o una prueba P_2 de B, podemos construir una prueba P de $A \vee B$. La regla de eliminación es un poco más confusa, ya que requiere dos razonamientos hipotéticos. Primero asumimos temporalmente A y después B, llegando en ambos casos a C. A partir de ello y de $A \vee B$ podemos inferir directamente C.

- Doble implicación:

$$\frac{\begin{array}{c}\overline{A}^{\,1}\\ \vdots \\ B\end{array} \qquad \begin{array}{c}\overline{B}^{\,1}\\ \vdots \\ A\end{array}}{A \leftrightarrow B}\,1\leftrightarrow\mathrm{I} \qquad \frac{A \leftrightarrow B \qquad A}{B}\,\leftrightarrow\mathrm{E}_i \qquad \frac{A \leftrightarrow B \qquad B}{A}\,\leftrightarrow\mathrm{E}_d$$

La regla de introducción expresa que si mediante razonamientos hipotéticos podemos construir una prueba de B a partir de A y viceversa, entonces tenemos una prueba de $A \leftrightarrow B$. Las reglas de eliminación funcionan como *Modus Ponens* pero en ambas direcciones, es por eso que también se las conoce como "Modus Ponens de la doble implicación" y "Modus Ponens Reverso".

- Verdad y falsedad lógica:

$$\frac{}{\top}\,\top\mathrm{I} \qquad \frac{\bot}{A}\,\bot\mathrm{E}$$

Siempre podemos introducir la verdad lógica, pues es trivialmente cierta. Respecto a la falsedad lógica, podemos eliminarla infiriendo cualquier fórmula A a partir de ella. Esta regla también es conocida como *ex falso*, refieriendose a la expresión latina "*ex falso sequitur quodlibet*" que significa "cualquier cosa sigue de la falsedad".

Observación 1.14. Notemos que si definimos $\neg A$ como $A \to \bot$, entonces las reglas de inferencia de la negación no son más que un caso concreto de las reglas de la implicación. Podemos pensar que $\neg A$ expresa que "si A es cierto entonces los cerdos vuelan" donde "los cerdos vuelan" se representa con $\bot$.

En la deducción natural, al no haber axiomas, las demostraciones parten directamente de una serie de hipótesis. Es decir, tenemos un conjunto de hipótesis $\{B, C, \dots\}$ y una conclusión A y la prueba lo que muestra es que A es implicación lógica de $B, C, \dots$. Este sistema en el que podemos ver las pruebas en terminos computacionales lo llamamos **lógica intuicionista**.

Para llegar al sistema lógico más usual, nos falta añadir una forma de razonamiento muy natural en las matemáticas: *la demostración por contradicción o reductio ad absurdium* (RAA). Se expresa mediante el esquema siguiente:

$$\dfrac{\dfrac{}{\neg A}\,1}{\begin{array}{c}\vdots\\ \bot\end{array}}\,1\ \text{RAA}$$

Esta herramienta no encaja en la lógica intuicionista, puesto que suponiendo $\neg A$ debemos llegar a producir una prueba de A. Denotaremos como L_C el sistema L_I al que hemos añadido la regla de inferencia RAA.

La consecuencia más importante de esta regla es el *principio del tercero excluido* que implica que una fórmula es verdadera o lo es su negación: $A \vee \neg A$ para A fórmula cualquiera en L_C.

Proposición 1.15. *Sea A fórmula de L_I. Entonces existe una prueba de $A \vee \neg A$ usando la demostración por contradicción.*

Demostración. Una prueba utilizando deducción natural y RAA podría ser la siguiente:

$$\dfrac{\dfrac{\dfrac{}{\neg(A\vee\neg A)}\,2 \qquad \dfrac{\dfrac{}{A}\,1}{A\vee\neg A}}{\dfrac{\bot}{\neg A}\,1\ \text{RAA}}\,\neg\text{E}}{\dfrac{\dfrac{}{\neg(A\vee\neg A)}\,2 \qquad \dfrac{\neg A}{A\vee\neg A}}{\dfrac{\bot}{A\vee\neg A}\,2\ \text{RAA}}}$$

□

De hecho, utilizando el principio del tercero excluido con la eliminación de la disyunción y *ex falso* se puede obtener la prueba por contradicción.

Proposición 1.16. *Sea A fórmula de L_I. Si $A \vee \neg A$ es demostrable y existe una prueba de $\bot$ a partir de $\neg A$, entonces existe una prueba de A con el esquema de la demostración por contradicción.*

Demostración. Basta considerar la siguiente prueba:

$$\dfrac{A\vee\neg A \qquad \dfrac{}{A}\,1 \qquad \dfrac{\dfrac{}{\neg A}\,1}{\begin{array}{c}\vdots\\ \bot\end{array}}}{A}\,\begin{array}{l}\bot\text{E}\\ 1\vee\text{E}\end{array}$$

□

Otro resultado interesante es la equivalencia de RAA con $\neg\neg A \leftrightarrow A$. La implicación de derecha a izquierda es intuicionista, la otra es conocida como la eliminación de la doble negación.

Proposición 1.17. *Sea A fórmula de L_I. Existe una prueba de A con el esquema de la demostración por contradicción si y solo si $\neg\neg A \leftrightarrow A$ es demostrable.*

Demostración. Veamos ambas implicaciones.

($\Leftarrow$) Supongamos que tenemos una prueba de $\bot$ a partir de $\neg A$, entonces:

$$\dfrac{\neg\neg A\leftrightarrow A \qquad \dfrac{\dfrac{}{\neg A}\,1}{\begin{array}{c}\vdots\\ \bot\end{array}}}{\neg\neg A}\,\begin{array}{l}\bot\text{E}\\ 1\leftrightarrow\text{E}\end{array}$$
$$A$$

($\Rightarrow$) Basta utilizar *ex falso* y la introducción de la doble implicación.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\neg A}\,3 \quad \dfrac{}{\neg A}\,1}{\bot}\,\neg\mathrm{E}}{A}\,1\,\mathrm{RAA} \qquad \dfrac{\dfrac{\dfrac{}{\neg A}\,2 \quad \dfrac{}{A}\,3}{\bot}\,\neg\mathrm{E}}{\neg\neg A}\,2\,\neg\mathrm{I}}{\neg\neg A \leftrightarrow A}\,3\,\leftrightarrow\mathrm{I}$$

□

1.3. Semántica de la lógica proposicional

La forma clásica de pensar sobre las variables es que estas pueden ser ciertas o falsas. Las pruebas nos dicen entonces qué fórmulas *tienen que ser* ciertas siempre, sin importar el valor de las variables. Por ejemplo si a partir de las hipótesis A, B podemos construir una prueba de C, $A \wedge B \to C$ es siempre cierto.

Para poder darle un significado debemos salir del sistema en sí y dar un valor a lo que es verdadero o falso. Esto es una noción *semántica* puesto que dota de significado a las fórmulas y pruebas, que no son más que estructuras *sintácticas*.

Si nos referimos a la sintaxis, podemos hacernos diversas preguntas:

- Dado un conjunto de hipótesis Γ y una fórmula A, ¿podemos derivar A de Γ?
- ¿Cuál es el conjunto de fórmulas que podemos derivar de Γ?
- ¿Cuál es el conjunto de hipótesis necesario para derivar A?

Respecto a la semántica, las preguntas son distintas:

- Dada una elección de valores para las variables que intervienen en la fórmula A, ¿es A cierta o falsa?
- ¿Existe una elección de valores que haga A cierta?
- ¿Qué elecciones de valores hacen a A cierta?

Denotaremos los valores de verdad por **V** y **F** para "verdadero" y "falso", respectivamente. Vamos a asumir también el razonamiento clásico ($S = L_C$), es decir, el *principio del tercero excluido*, por tanto toda proposición es verdadera o falsa y no ambas.

Definición 1.18. Sea $\mathbb{V}$ un conjunto de variables proposicionales, una **evaluación de verdad** es una función $v\colon \mathbb{V} \to \{\mathbf{V}, \mathbf{F}\}$.

A partir de una evaluación de verdad, podemos dar valores de verdad a todas las fórmulas que dependan de ese conjunto de variables.

Definición 1.19. Dada una evaluación de verdad v, podemos extenderla a una **función evaluación** $\bar{v}\colon \mathcal{F}_{\mathbb{V}} \to \{\mathbf{V}, \mathbf{F}\}$, donde $\mathcal{F}_{\mathbb{V}}$ es el conjunto de fórmulas dependientes solo de las variables proposicionales de $\mathbb{V}$. Se define recursivamente, si $A, B \in \mathcal{F}_{\mathbb{V}}$:

- $\bar{v}(\top) \coloneqq \mathbf{V}$
- $\bar{v}(\bot) \coloneqq \mathbf{F}$
- $\bar{v}(p) \coloneqq v(p) \quad \forall p \in \mathbb{V}$
- $\bar{v}(\neg A) \coloneqq \begin{cases} \mathbf{V} & \text{si } \bar{v}(A) = \mathbf{F} \\ \mathbf{F} & \text{si } \bar{v}(A) = \mathbf{V} \end{cases}$
- $\bar{v}(A \wedge B) \coloneqq \begin{cases} \mathbf{V} & \text{si } \bar{v}(A) = \bar{v}(B) = \mathbf{V} \\ \mathbf{F} & \text{en otro caso} \end{cases}$
- $\bar{v}(A \vee B) \coloneqq \begin{cases} \mathbf{F} & \text{si } \bar{v}(A) = \bar{v}(B) = \mathbf{F} \\ \mathbf{V} & \text{en otro caso} \end{cases}$

- $\bar{v}(A \to B) := \begin{cases} \mathbf{F} & \text{si } \bar{v}(A) = \mathbf{V} \text{ y } \bar{v}(B) = \mathbf{F} \\ \mathbf{V} & \text{en otro caso} \end{cases}$
- $\bar{v}(A \leftrightarrow B) := \begin{cases} \mathbf{V} & \text{si } \bar{v}(A) = \bar{v}(B) \\ \mathbf{F} & \text{en otro caso} \end{cases}$

Con esto hemos respondido a la primera pregunta, ¿pero cómo sabemos si existe una evaluación de verdad que haga A verdadero? Esta pregunta nos hace pensar en todas las posibles evaluaciones de verdad. Claramente el número depende de la cantidad de variables en $\mathbb{V}$ a considerar y al tener 2 posibles valores de verdad ($\mathbf{V}, \mathbf{F}$), se tienen $2^{|\mathbb{V}|}$ posibles evaluaciones de verdad. Por lo tanto, el cálculo se vuelve exponencialmente más costoso.

Para averiguar si una fórmula llega a ser verdadera, utilizamos una **tabla de verdad** con $2^{|\mathbb{V}|}$ filas, poniendo a la izquierda todas las posibles evaluaciones de verdad para las variables y a la derecha, el valor de verdad de la fórmula en cada una.

Ejemplo 1.20. Podemos construir tablas de verdad para resumir la semántica de los conectores lógicos de nuestro sistema:

A	$\neg A$
F	**V**
V	**F**

A	B	$A \wedge B$
F	**F**	**F**
F	**V**	**F**
V	**F**	**F**
V	**V**	**V**

A	B	$A \vee B$
F	**F**	**F**
F	**V**	**V**
V	**F**	**V**
V	**V**	**V**

A	B	$A \to B$
F	**F**	**V**
F	**V**	**V**
V	**F**	**F**
V	**V**	**V**

A	B	$A \leftrightarrow B$
F	**F**	**V**
F	**V**	**F**
V	**F**	**F**
V	**V**	**V**

Para las fórmulas compuestas puede ser más cómodo añadir columnas intermedias para las subfórmulas, por ejemplo:

A	B	C	$A \to B$	$B \to C$	$(A \to B) \vee (B \to C)$
F	**F**	**F**	**V**	**V**	**V**
F	**F**	**V**	**V**	**V**	**V**
F	**V**	**F**	**V**	**F**	**V**
F	**V**	**V**	**V**	**V**	**V**
V	**F**	**F**	**F**	**V**	**V**
V	**F**	**V**	**F**	**V**	**V**
V	**V**	**F**	**V**	**F**	**V**
V	**V**	**V**	**V**	**V**	**V**

En una tabla de verdad podemos ver a simple vista qué evaluaciones de verdad hacen a la fórmula verdadera. Observamos que en la tabla anterior, la ultima columna siempre tiene el valor **V**, es decir, la fórmula *siempre* es verdadera.

Definición 1.21. Sea A una fórmula, si $\bar{v}(A) = \mathbf{V}\ \forall v$, siendo v una evaluación de verdad de las variables que intervienen en A, decimos que A es **válida** o que es una **tautología**. Lo denotaremos como $\vDash_S A$, omitiendo el sistema formal S cuando esté claro.

Proposición 1.22. *Los axiomas ($\to 1$), ($\to 2$) y ($\to 3$) son tautologías.*

Demostración. Basta construir las tablas de verdad de los axiomas.

A	B	$B \to A$	$A \to (B \to A)$
F	**F**	**V**	**V**
F	**V**	**F**	**V**
V	**F**	**V**	**V**
V	**V**	**V**	**V**

A	B	C	$A \to (B \to C)$	$(A \to B) \to (A \to C)$	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
F	**F**	**F**	**V**	**V**	**V**
F	**F**	**V**	**V**	**V**	**V**
F	**V**	**F**	**V**	**V**	**V**
F	**V**	**V**	**V**	**V**	**V**
V	**F**	**F**	**V**	**V**	**V**
V	**F**	**V**	**V**	**V**	**V**
V	**V**	**F**	**F**	**F**	**V**
V	**V**	**V**	**V**	**V**	**V**

A	B	$\neg B \to \neg A$	$(\neg B \to A) \to B$	$(\neg B \to \neg A) \to (\neg B \to A) \to B$
F	**F**	**V**	**V**	**V**
F	**V**	**V**	**V**	**V**
V	**F**	**F**	**F**	**V**
V	**V**	**V**	**V**	**V**

Comprobamos que todas las posibles evaluaciones de verdad hacen a los axiomas verdaderos. □

También es sencillo comprobar que la regla de inferencia Modus Ponens preserva tautologías.

Proposición 1.23. *Si A y $A \to B$ son tautologías, entonces B también lo es.*

Demostración. Supongamos que $\exists v$ evaluación de verdad de las variables que intervienen en A y en B tal que $\bar{v}(B) = \mathbf{F}$. Como A es una tautología, $\bar{v}(A) = \mathbf{V}$. Por la definición recursiva de $\bar{v}$, entonces tendríamos $\bar{v}(A \to B) = \mathbf{F}$, que contradice que $A \to B$ sea tautología. □

1.4. Solidez y completitud

Que una fórmula sea demostrable es una noción sintáctica, mientras que ser una tautología pertenece al ámbito semántico. Sin embargo parece intuitivo que ambas coincidan, puesto que expresan el hecho de que la fórmula *tiene que ser* verdadera y uno esperaría poder derivar las fórmulas válidas.

Definición 1.24. Sea S un sistema formal en el que consideramos los valores de verdad **V** y **F**. Decimos que S es **sólido** (*sound* en inglés) si toda fórmula demostrable es una tautología, es decir, $\vdash A \implies \vDash A \ \forall A$ fórmula de S.

Recíprocamente, decimos que S es **completo** si toda fórmula que es una tautología es demostrable, es decir, $\vDash A \implies \vdash A \ \forall A$ fórmula de S.

Estas nociones las podemos extender a un caso más general, cuando utilizamos un conjunto de hipótesis. Ya hemos definido qué significa que una fórmula se derive de un conjunto de hipótesis, veamos su contraparte semántica:

Definición 1.25. Sea S un sistema formal en el que consideramos los valores de verdad **V** y **F**, A una fórmula y Γ un conjunto de fórmulas de S (hipótesis). Decimos que A es **consecuencia lógica** de Γ si toda v asignación de verdad que hace verdadera toda fórmula en Γ, también hace A verdadera: $\forall v$ evaluación de verdad $(v(B) = \mathbf{V} \ \forall B \in \Gamma) \implies v(A) = \mathbf{V}$. Lo denotamos $\Gamma \vDash A$.

Definición 1.26. Sea S un sistema formal y Γ un conjunto de fórmulas de S (hipótesis). Decimos que S junto a las hipótesis de Γ es **sólido** (*sound* en inglés) si toda fórmula derivable de Γ es consecuencia lógica de Γ, es decir, $\Gamma \vdash A \implies \Gamma \vDash A \ \forall A$ fórmula de S.

Recíprocamente, decimos que S junto a las hipótesis de Γ es **completo** si toda fórmula consecuencia lógica de Γ es derivable de Γ, es decir, $\Gamma \vDash A \implies \Gamma \vdash A \ \forall A$ fórmula de S.

Con las propiedades vistas en los apartados anteriores, es sencillo demostrar que el sistema de la lógica proposicional es sólido.

Teorema 1.27 (Solidez de L_P)**.** *El sistema de la lógica proposicional L_P es sólido.*

Demostración. Basta aplicar la Proposición 1.23 a la Proposición 1.22, de forma que si existe una prueba para A usando los axiomas y Modus Ponens, A es una tautología. □

Para demostrar la completitud de L_P vamos a necesitar el siguiente lema:

Lema 1.28. *Sea $S = L_P$, A una fórmula y $B_1, \dots, B_k$ las variables proposicionales que aparecen en A. Para una evaluación de verdad v de estas variables, consideramos B'_j como B_j si $v(B_j) = \boldsymbol{V}$ y B'_j como $\neg B_j$ si $v(B_j) = \boldsymbol{F}$ ($j = 1, \dots, k$). Consideramos A' como A si $\bar{v}(A) = \boldsymbol{V}$ y A' como $\neg A$ si $\bar{v}(A) = \boldsymbol{F}$. Entonces se tiene que $B'_1, \dots, B'_k \vdash A'$*

Demostración. Por inducción en el número n de apariciones de $\neg$ y $\to$ en A. Si $n = 0$, $A = B_1$ es una variable proposicional y es trivialmente cierto que $B_1 \vdash B_1$ y que $\neg B_1 \vdash \neg B_1$. Asumimos que es cierto para $j < n$. Tenemos dos casos:

1. A es $\neg C$ para alguna fórmula C. Entonces el número de apariciones de $\neg$ y $\to$ en C es menor que n.

   a) Si $\bar{v}(C) = \mathbf{V}$, entonces $\bar{v}(A) = \bar{v}(\neg C) = \mathbf{F}$. Por lo tanto, $C' = C$ y $A' = \neg A = \neg\neg C$. Por hipótesis de inducción sobre C, $B'_1, \dots, B'_k \vdash C$. Por la Proposición 1.11-4, $\vdash C \to \neg\neg C$ y aplicando Modus Ponens se tiene que $B'_1, \dots, B'_k \vdash \neg\neg C$.

   b) Si $\bar{v}(C) = \mathbf{F}$, entonces $\bar{v}(A) = \bar{v}(\neg C) = \mathbf{V}$. Por lo tanto, $C' = \neg C$ y $A' = A = \neg C$. Por hipótesis de inducción sobre C, $B'_1, \dots, B'_k \vdash \neg C$.

2. A es $C \to D$ para algunas fórmulas C y D. Entonces el el número de apariciones de $\neg$ y $\to$ en C y en D es menor que n. Por hipótesis de inducción $B'_1, \dots, B'_k \vdash C'$ y $B'_1, \dots, B'_k \vdash D'$.

   a) Si $\bar{v}(C) = \mathbf{F}$, entonces $\bar{v}(A) = \bar{v}(C \to D) = \mathbf{V}$. Por lo tanto, $C' = \neg C$ y $A' = A$. Por hipótesis de inducción $B'_1, \dots, B'_k \vdash \neg C$ y por la Proposición 1.11-5, $\vdash \neg C \to (C \to D)$. Aplicando Modus Ponens se tiene que $B'_1, \dots, B'_k \vdash C \to D$.

   b) Si $\bar{v}(D) = \mathbf{V}$, entonces $\bar{v}(A) = \bar{v}(C \to D) = \mathbf{V}$. Por lo tanto, $D' = D$ y $A' = A$. Por hipótesis de inducción $B'_1, \dots, B'_k \vdash D$ y por el Axioma ($\to$ 1), $\vdash D \to (C \to D)$. Aplicando Modus Ponens se tiene que $B'_1, \dots, B'_k \vdash C \to D$.

   c) Si $\bar{v}(C) = \mathbf{V}$ y $\bar{v}(D) = \mathbf{F}$, entonces $\bar{v}(A) = \bar{v}(C \to D) = \mathbf{F}$. Por lo tanto, $C' = C$, $D' = \neg D$ y $A' = \neg A = \neg(C \to D)$. Por hipótesis de inducción $B'_1, \dots, B'_k \vdash C$ y $B'_1, \dots, B'_k \vdash \neg D$. y por la Proposición 1.11-8, $\vdash C \to (\neg D \to \neg(C \to D))$. Aplicando Modus Ponens dos veces se tiene que $B'_1, \dots, B'_k \vdash \neg(C \to D)$.

□

Observación 1.29. El Lema 1.28 en esencia nos dice que podemos interpretar cada fila de una tabla de verdad como una prueba a partir de las variables y fórmulas que intervienen; considerándolas sin negar si toman el valor **V** y negadas si toman el valor **F**.

Teorema 1.30 (Completitud de L_P)**.** *El sistema de la lógica proposicional L_P es completo.*

Demostración. (Kalmár, 1935) Sea A una tautología y $B_1, \dots, B_k$ las variables proposicionales que aparecen en A. Para una evaluación de verdad v cualquiera tenemos por el Lema 1.28, $B'_1, \dots, B'_k \vdash A$ ya que $\bar{v}(A) = \mathbf{V}$. Cuando B'_k toma el valor **V** se tiene que $B'_1, \dots, B_k \vdash A$, y cuando B'_k toma el valor **F** se tiene que $B'_1, \dots, \neg B_k \vdash A$. Por el Teorema de la Deducción (1.8), tenemos que $B'_1, \dots, B'_{k-1} \vdash B_k \to A$ y $B'_1, \dots, B'_{k-1} \vdash \neg B_k \to A$. Por la Proposición 1.11-9, $\vdash (B_k \to A) \to ((\neg B_k \to A) \to A)$ y aplicando Modus Ponens dos veces, $B'_1, \dots, B'_{k-1} \vdash A$. Análogamente podemos eliminar B'_{k-1} considerando que toma los valores **V** y **F**. Tras k pasos, obtenemos que $\vdash A$.

□

La principal consecuencia de los Teoremas 1.27 y 1.30 es que la noción de que las fórmulas verdaderas son exactamente las demostrables es cierta. No existen fórmulas verdaderas que no se puedan demostrar ni fórmulas demostrables que sean falsas.

Definición 1.31. Sea S un sistema formal. Decimos que S es **consistente** si no existe ninguna fórmula A en S de forma que tanto A como $\neg A$ sean demostrables.

Puesto que la negación de una tautología no es una tautología, es imposible que tanto una fórmula como su negación tengan una prueba en L_P. Se tiene por tanto el siguiente Corolario:

Corolario 1.32 (Consistencia de L_P)**.** *El sistema de la lógica proposicional L_P es consistente.*

Capítulo 2

Sistemas de primer orden

Si intentamos trabajar en un contexto más general, nos daremos cuenta de que hay conclusiones lógicas que no podemos inferir en la lógica proposicional. Por ejemplo, si "Existe un gato azul" y "Los gatos son animales" uno infiere que "Existe un animal azul", sin embargo las herramientas anteriores no son suficientes para ello.

Notemos primero que ahora estamos hablando de propiedades de los elementos, por ello tiene sentido introducir un nuevo concepto: las variables y los predicados y funciones sobre ellas. Denotaremos a las variables por las letras $x, y, z, \ldots$. Los predicados son fórmulas en las que intervienen variables, por ejemplo $P(x)$ representa que en el predicado P interviene la variable x. Las funciones actúan de la manera natural, toman un número de elementos del dominio $x_1, \ldots, x_n$ y devuelven un resultado, que también pertenece al dominio $f(x_1, \ldots, x_n)$.

Ejemplo 2.1. Si consideramos el sistema de los números naturales junto a las posibles afirmaciones y funciones sobre ellos, podemos usar predicados y relaciones entre ellos, que ademas podemos componer con conectores lógicos:

- $(x+y<z) \wedge (x+y<-z)$, aquí $x+y$ supone notación infija para representar $suma(x,y)$
- $par(x) \vee impar(x)$
- $\neg(cuadrado(x+y)=z)$

Los sistemas que nos permite hacer afirmaciones generales sobre estos predicados son los **sistemas de primer orden**. Para ello, utilizamos cuantificadores:

- El cuantificador universal $\forall$, seguido de una variable x y un predicado o relación. Representa "Para todo x", es decir, que todo valor de x cumple la propiedad que sigue.
- El cuantificador existencial $\exists$, seguido de una variable x y un predicado o relación. Representa "Existe un x", es decir, que hay algún valor de x que cumple la propiedad que sigue.

Notemos que $\exists x\, P(x) \equiv \neg\forall x\, \neg P(x)$.

Definición 2.2. Un **sistema de primer orden** L es un sistema formal donde el alfabeto contiene un conjunto de símbolos de variables, constantes, funciones, predicados y relaciones. Un término se define recursivamente como:

1. Las variables y constantes son términos.
2. Si f es un símbolo función y $t_1, \ldots, t_n$ son términos, $f(t_1, \ldots, t_n)$ también lo es.

Así, las fórmulas de L son:

1. Si P es un símbolo predicado y t es un término, $P(t)$ es una fórmula.

2. Si R es un símbolo relación y $t_1, \ldots, t_n$ son términos, $R(t_1, \ldots, t_n)$ es una fórmula.
3. La combinación recursiva de fórmulas con conectores lógicos es una fórmula.

Definición 2.3. Si tenemos un fórmula $\forall x\, P(x)$ o $\exists x\, P(x)$ y P involucra a x, decimos que la variable está **ligada** por el cuantificador. En el caso de que no la involucre, decimos que es **libre**. Las fórmulas que no contienen variables libres se dicen **cerradas**.

Observación 2.4. Las variables ligadas son mudas, ya que la expresión no habla de una variable concreta. Así $\forall x\ par(x) \vee impar(x)$ es la misma fórmula que $\forall y\ par(y) \vee impar(y)$.

Los cuantificadores son conectores y como tales, tienen sus reglas de inferencia que permiten introducirlos y eliminarlos:

- Para el cuantificador universal, la regla de introducción se conoce como generalización, ya que si no suponemos nada sobre la variable, podemos afirmar que se cumple para cualquier variable. Por ello x no puede ser libre en niguna hipótesis sin cancelar. La regla de eliminación representa la elección de un término concreto para ver que este cumple la fórmula, por ello t puede ser cualquier término que no cause conflicto con las variables ligadas de A.

$$\frac{A(x)}{\forall y\, A(y)}\ \forall \mathrm{I} \qquad \frac{\forall x\, A(x)}{A(t)}\ \forall \mathrm{E}$$

- Para el cuantificador existencial, en la regla de introducción a partir de un término cumpliendo A, podemos inferir que $\exists x\, A(x)$, por ello t puede ser cualquier término que no cause conflicto con las variables ligadas de A. En la regla de eliminación si $\exists x\, A(x)$ y tenemos prueba de B a partir de $A(y)$, podemos inferir directamente B, pero y no puede ser libre en B ni en niguna hipótesis sin cancelar.

$$\frac{A(t)}{\exists x\, A(x)}\ \exists \mathrm{I} \qquad \frac{\exists x\, A(x) \qquad \begin{array}{c}\overline{A(y)}^{\,1}\\ \vdots \\ B\end{array}}{B}\ 1\ \exists \mathrm{E}$$

Adicionalmente, en la lógica simbólica usamos expresiones de igualdad, por ejemplo cuando decimos que "El asesino es el mayordomo" o que "$2 * 3 = 6$", donde ambas descripciones se refieren al mismo objeto. Como esta noción es aplicable a cualquier contexto de objetos, cae en el ámbito de la lógica.

Axiomáticamente, asumimos que la igualdad cumple las siguientes 3 propiedades:

1. Reflexividad: $t = t$ para cualquier término t
2. Simetría: si $t = s$, entonces $s = t$
3. Transitividad: si $r = s$ y $s = t$, entonces $r = t$

Sin embargo, no son suficientes para caracterizarla. Si dos expresiones denotan al mismo objeto, deberíamos poder sustituir el uno por el otro en cualquier expresión. Así utilizaremos la convención de que si r es una expresión cualquiera, $r(x)$ indica que la variable x puede ocurrir en r. Entonces si s es otro término cualquiera, $r(s)$ denota el resultado de sustituir x por s en r. La regla de sustitución es por tanto: si $s = t$, entonces $r(s) = r(t)$. Con las fórmulas ocurre análogamente.

Con esto en mente, las reglas de inferencia para la igualdad quedan:

$$\frac{}{t = t}\ \mathrm{refl} \qquad \frac{s = t}{t = s}\ \mathrm{sim} \qquad \frac{r = s \qquad s = t}{r = t}\ \mathrm{trans}$$

$$\frac{s = t}{r(s) = r(t)}\ \text{subst} \qquad \frac{s = t \qquad P(s)}{P(t)}\ \text{subst}$$

Llamamos **lógica de primer orden** (L_{PO}) al sistema de primer orden construido a partir de L_P junto al cuantificador universal, sus reglas y el siguiente axioma:

($\forall 1$) $\forall x\,(A \to B) \to (A \to \forall x\,B)$, donde x no es libre en A

En L_{PO} consideramos únicamente los valores de verdad **V** y **F**.

2.1. Semántica de la lógica de primer orden

Tal y como comentamos en el Apartado 1.3, podemos hacer la misma distinción entre sintaxis y semántica con la lógica de primer orden.

Consideremos como dominio $\mathbb{N}$ con los símbolos $0, 1, 2, \ldots$, las funciones *suma* y *mul* y los predicados $par, primo, le$. La fórmula $\forall y\ \ le(0, y)$ es cierta en este ejemplo, si interpretamos le como la relación menor-o-igual en los números naturales. Sin embargo si consideramos $\mathbb{Z}$, la misma fórmula es falsa. También podríamos considerar (de manera perversa) el predicado $le(x, y)$ como la relación "x es mayor que y" y la fórmula ya no sería cierta.

Esto nos dice que la veracidad de las fórmulas de primer orden puede depender de la interpretación de los cuantificadores y relaciones del lenguaje. Por otro lado también hay fórmulas que son verdaderas en cualquier interpretación: por ejemplo, $\forall y\ \ (le(0, y) \to le(0, y))$. Esto es análogo a las *tautologías* que ya hemos visto en la lógica proposicional.

Esta analogía se puede extender más: un "modelo" será el equivalente a una evaluación de verdad. De la misma forma que escogiendo una evaluación de verdad nos permite asignar valores de verdad a todas las fórmulas de la lógica proposicional; escogiendo un modelo podremos asignar valores de verdad a todas las fórmulas de un sistema de primer orden.

2.2. Interpretaciones y modelos

Los símbolos del sistema de ejemplo -*par*, *suma*, 0- tienen nombres que nos sugieren su comportamiento: para qué elementos *par* "debería" ser cierto y para cuales falso. Consideremos otro sistema de primer orden con símbolos *elegante* y *alto* en el dominio $\mathbb{N}$, ¿es cierto que $\forall x\,(elegante(x) \to alto(x))$?

La respuesta es claramente que no tenemos suficiente información como para decirlo. No hay un significado claro para los predicados en los números naturales. Esto motiva la siguiente definición:

Definición 2.5. Una **interpretación** de un sistema de primer orden sobre un conjunto de elementos D es la interpretación de cada uno de sus símbolos sobre D.

- Una interpretación sobre D de un símbolo predicado P es un conjunto $P_I \subseteq D$.
- Una interpretación sobre D de un símbolo relación n-aria R es un conjunto $R_I \subseteq D^n$.
- Una interpretación sobre D de un símbolo constante c es un elemento $c_I \in D$.
- Una interpretación sobre D de un símbolo función n-aria f es una función $f_I : D^n \to D$.

Observación 2.6. Es importante enfatizar la diferencia entre los elementos sintácticos y sus interpretaciones semánticas. El primero es un símbolo que se relaciona con más símbolos y no tiene significado en sí. Por tanto no tiene sentido escribir $primo(5)$, donde *primo* es un predicado y 5 es un número natural, ya que el primero es un elemento sintáctico y el segundo, un elemento del dominio. En ocasiones la distinción no es tan clara, al usar símbolos constantes como $0, 1, 2$; sin embargo sigue habiendo una diferencia fundamental entre los elementos del dominio y los símbolos que utilizamos para representarlos.

Es habitual que para cada elemento a del dominio, introduzcamos un símbolo constante $\bar{a}$ que se interprete como a. De esta manera, sí que tiene sentido la expresión $primo(\bar{5})$.

Definición 2.7. Un **modelo** $\mathcal{M}$ de un sistema de primer orden L es el par formado por un conjunto de elementos D que llamamos **dominio** y una interpretación de L sobre D.

Definición 2.8. La **interpretación de un término** en un modelo $\mathcal{M}$ es el elemento resultante de la interpretación recursiva de sus símbolos. Si el término es de la forma $f(t_1,\dots,t_n)$, se interpretan primero los términos $t_1,\dots,t_n$ y después se aplica la interpretación de f a estos.

La **interpretación de una relación** $R(x_1,\dots,x_n)$ en un modelo $\mathcal{M}$ es **V** si la interpretación de los términos $x_1,\dots,x_n$ en $\mathcal{M}$ está en R_I y **F** en caso contrario. Para $n=1$, tenemos la interpretación de un predicado.

La **interpretación de una fórmula cerrada** en un modelo $\mathcal{M}$ es el valor de verdad resultante de la interpretación recursiva de sus términos, relaciones, cuantificadores y conectores lógicos. Estos últimos se evalúan recursivamente como la función evaluación (Definición 1.19). Denotamos $\mathcal{M} \vDash A$ si A se evalua a **V** en $\mathcal{M}$, y $\mathcal{M} \nvDash A$ si A se evalua a **F**. (Leemos el símbolo $\vDash$ como "modela" o "valida".)

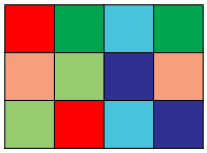
Respecto a los cuantificadores, decimos que $\mathcal{M} \vDash \exists x\, A$ cuando hay un elemento a del dominio de $\mathcal{M}$ tal que $\mathcal{M} \vDash A[\bar{a}/x]$, donde la notación $A[\bar{a}/x]$ indica que se sustituye cada aparición de x en A por el símbolo $\bar{a}$. Análogamente, decimos que $\mathcal{M} \vDash \forall x\, A$ cuando para todo elemento a del dominio de $\mathcal{M}$ se tiene que $\mathcal{M} \vDash A[\bar{a}/x]$.

Ejemplo 2.9. Podemos considerar un sistema con un símbolo de relación $\leq$ y un símbolo de función binaria $+$ con notación infija. Nos preguntamos ahora por el valor de verdad de $\forall x \exists y\,(x \leq y)$ y de $\exists x \forall y\,(x \leq y)$:

1. Si el modelo $\mathcal{M}$ tiene el dominio $\mathbb{N}$ con la suma y la relación $\leq$ sobre los naturales, entonces ambas son ciertas. La primera porque todo elemento es menor o igual que él mismo ; la segunda porque existe un elemento (0) menor o igual que todos los demás.

2. Si el modelo $\mathcal{M}$ tiene el dominio $\mathbb{Z}$ con la suma y la relación $\leq$ sobre los enteros, entonces la segunda ya no es cierta, ya que para cada elemento hay también uno menor ($\mathbb{Z}$ no está acotado inferiormente).

Los modelos a utilizar no tienen por qué ser solo conjuntos numéricos. Consideremos ahora un sistema con los predicados *verde*, *azul*, *rojo*, *claro* y *oscuro*, interpretados de la manera obvia. Además tenemos las relaciones $\mathit{misma\text{-}fila}(x,y)$, $\mathit{misma\text{-}columna}(x,y)$, $\mathit{mismo\text{-}color}(x,y)$ y $\mathit{misma\text{-}intensidad}(x,y)$ que también tienen la interpretación evidente. Tomamos el modelo con el siguiente dominio:



Podemos comprobar por ejemplo que:

1. $\mathcal{M} \vDash \forall x\ (\mathit{verde}(x) \to (\exists y\ \mathit{misma\text{-}columna}(x,y) \wedge \mathit{rojo}(y)))$, ya que todo cuadro verde tiene otro rojo en la misma columna.

2. $\mathcal{M} \vDash \exists x \forall y\ (\mathit{misma\text{-}columna}(x,y) \to \mathit{mismo\text{-}color}(x,y))$, ya que existe una columna con todos los cuadros del mismo color.

3. $\mathcal{M} \vDash \forall x\ (\exists y\ \neg \mathit{mismo\text{-}color}(x,y) \wedge \neg \mathit{misma\text{-}intensidad}(x,y))$, ya que para cada cuadro hay otro de distinto color y distinta intensidad.

Definición 2.10. Sea L un sistema de primer orden en el que consideramos los valores de verdad **V** y **F** y A una fórmula, si $\mathcal{M} \vDash A\ \forall \mathcal{M}$, siendo $\mathcal{M}$ un modelo de L, decimos que A es **válida** o que es una **tautología**. Lo denotaremos como $\vDash_L A$, omitiendo el sistema L cuando esté claro.

Definición 2.11. Sea L un sistema de primer orden en el que consideramos los valores de verdad **V** y **F**, A una fórmula y Γ un conjunto de fórmulas de L (hipótesis). Decimos que A es **consecuencia lógica** de Γ si todo modelo de Γ es también modelo de A. Lo denotamos $\Gamma \vDash A$.

Observación 2.12. Notemos que estamos redefiniendo los conceptos de tautología y consecuencia lógica de las Definiciones 1.21 y 1.25 para que coincidan con la noción de verdad en los sistemas de primer orden. Con esta definición, una fórmula no cerrada A es válida si y solo si lo es su clausura universal, esto es, $\forall x_{i_1} \dots \forall x_{i_n} A$, donde $x_{i_1}, \dots, x_{i_n}$ son las variables libres de A.

En el caso de la lógica proposicional, vimos que comprobar si una fórmula era válida era algorítmico. Sin embargo, no sucede lo mismo en sistemas de primer orden. Para cada sistema existen infinitos modelos, con lo que la posible tabla de verdad sería infinitamente larga. Además, decidir si un enunciado universal como $\forall x\, P(x)$ es cierto en un modelo con dominio infinito puede requerir comprobar si P es cierto con una cantidad infinita de elementos.

Definición 2.13. Sea L un sistema de primer orden en el que consideramos los valores de verdad **V** y **F**. Decimos que L es **sólido** si toda fórmula demostrable es una tautología, es decir, $\vdash A \implies \vDash A$ $\forall A$ fórmula de S.

Recíprocamente, decimos que L es **completo** si toda fórmula que es una tautología es demostrable, es decir, $\vDash A \implies \vdash A$ $\forall A$ fórmula de L.

Observación 2.14. Pese a que en apariencia las definciones son las mismas que en la Definición 1.24, en realidad son distintas, puesto que aunque el concepto de *fórmula demostrable* es el mismo, hemos redefinido la noción de *tautología* en la Definición 2.10.

Veamos ahora que el sistema de la lógica de primer orden es sólido, siguiendo las mismas ideas que vimos para la lógica proposicional.

Teorema 2.15 (Solidez de L_{PO})**.** *El sistema de la lógica de primer orden L_{PO} es sólido.*

Demostración. Siguiendo la demostración del Teorema 1.27, basta comprobar que las reglas de introducción y eliminación del cuantificador universal preservan la verdad y que ($\forall$1) es válida, puesto que las tautologías de L_P son ciertas para cualquier interpretación de L_{PO}.

Supongamos que existe una interpretación en la que ($\forall$1) no es cierta. Entonces existe un modelo $\mathcal{M}$ donde $\mathcal{M} \vDash \forall x\,(A \to B)$ y $\mathcal{M} \nvDash (A \to \forall x\, B)$. Por tanto, $\mathcal{M} \vDash A$ y $\mathcal{M} \nvDash \forall x\, B$. Existe entonces un elemento a tal que $\mathcal{M} \nvDash B[\bar{a}/x]$. Por otro lado, sustituyendo en A y en $\forall x\,(A \to B)$, tenemos que $\mathcal{M} \vDash A[\bar{a}/x]$ y $\mathcal{M} \vDash (A \to B)[\bar{a}/x]$. Entonces por *Modus Ponens*, $\mathcal{M} \vDash B[\bar{a}/x]$ y llegamos a una contradicción. Por tanto ($\forall$1) es válida.

Supongamos que $\vDash A$. Si $\mathcal{M}$ es un modelo cualquiera, entonces $\mathcal{M} \vDash A[\bar{a}/x]$ para cualquier elemento a del dominio, es decir: $\mathcal{M} \vDash \forall x\, A$ y por tanto $\vDash \forall x\, A$ y ($\forall$I) preserva la verdad.

Respecto a ($\forall$E), si $\vDash \forall x\, A$, es claro que sustituyendo por un término t cualquiera se tiene que $\mathcal{M} \vDash A[t/x]$ para cualquier modelo y por tanto preserva la verdad. □

Para demostrar la completitud de L_{PO}, necesitaremos un par de lemas previos.

Definición 2.16. Sea $S = (\mathcal{A}, \Omega, I, Z)$ un sistema formal, Γ un conjunto de fórmulas de S y $S' = (\mathcal{A}, \Omega, I \cup \Gamma, Z)$. Decimos que Γ es **consistente** si S' es consistente.

Lema 2.17. *Si A es una fórmula de L_{PO} tal que $\Gamma \nvdash A$, entonces $\Gamma \cup \{\neg A\}$ es consistente.*

Demostración. Supongamos que $\Gamma \cup \{\neg A\}$ no es consistente, es decir, $\exists B$ fórmula de $L'_{PO} = (\mathcal{A}_{PO}, \Omega_{PO}, I_{PO} \cup \Gamma, Z_{PO})$ de forma que $\Gamma \cup \{\neg A\} \vdash B$ y $\Gamma \cup \{\neg A\} \vdash \neg B$. Por la Proposición 1.11-5[1], tenemos que $\Gamma \cup \{\neg A\} \vdash \neg B \to (B \to A)$ y aplicando dos veces *Modus Ponens* llegamos a que $\Gamma \cup \{\neg A\} \vdash A$. Aplicando el Teorema 1.8[2] a esto último, tenemos que $\Gamma \vdash \neg A \to A$.

[1] Podemos aplicar estas propiedades de L_P en L_{PO}, ya que las demostraciones solo involucran los Axiomas ($\to$ 1), ($\to$ 2), ($\to$ 3) y *Modus Ponens*.

[2] Ver la Nota 1.

Por otro lado, por $(\rightarrow 3)$ se tiene que $\Gamma \vdash (\neg A \rightarrow \neg A) \rightarrow ((\neg A \rightarrow A) \rightarrow A)$ y por el Ejemplo 1.4[3] $\Gamma \vdash \neg A \rightarrow \neg A$. Aplicando de nuevo *Modus Ponens* dos veces concluimos que $\Gamma \vdash A$, contradiciendo la hipótesis de que $\Gamma \nvdash A$. □

Lema 2.18. *En el sistema de la lógica de primer orden L_{PO} todo conjunto consistente de fórmulas tiene un modelo.*

Esquema de demostración. La demostración completa queda fuera del alcance de este trabajo. En esencia, lo que hacemos es expandir el sistema con una cantidad numerable de constantes nuevas $c_0, c_1, \ldots$, enumerar las fórmulas con una sola variable libre $A_0(x_{i_0}), A_1(x_{i_1}), \ldots$ y construir una sucesión creciente de sistemas formales consistentes a partir de ello. Este proceso sigue hasta llegar a un sistema completo, es decir, en el que $\vDash A$ o $\vDash \neg A$ para cualquier fórmula A. Por último, se construye una interpretación en la que el dominio son los términos cerrados del sistema completo. Este modelo también valida el sistema original, con lo que tenemos lo buscado. □

Teorema 2.19 (Teorema de Completitud de Gödel)**.** *El sistema de la lógica de primer orden L_{PO} es completo.*

Demostración. Supongamos que no existe ninguna prueba de A en L_{PO}. Entonces por el Lema 2.17 el conjunto $\{\neg A\}$ es consistente. Por tanto, por el Lema 2.18, existe un modelo $\mathcal{M}$ de $\{\neg A\}$. Pero en particular es un modelo de L_{PO} que no valida A, con lo que A no es una tautología. □

Análogamente a como hemos visto en la lógica proposicional, al caracterizar las fórmulas demostrables como las válidas se tiene el siguiente Corolario:

Corolario 2.20 (Consistencia de L_{PO})**.** *El sistema de la lógica de primer orden L_{PO} es consistente.*

Observación 2.21. A primera vista este hecho puede parecer contradictorio con el Teorema de Incompletitud de Gödel, que afirma que ninguna teoría matemática formal de primer orden que contenga los números naturales y la aritmética, es a la vez consistente y completa. Esa condición es la que marca la diferencia: la lógica de primer orden no tiene suficiente capacidad expresiva como para construir la aritmética de los naturales. El sistema al que se refiere Gödel añade infinitos axiomas que permiten construir demostraciones por inducción pero al mismo tiempo, vuelven al sistema incompleto o inconsistente.

2.3. Sistemas de orden superior

Tal y como se puede expandir la lógica proposicional a la lógica de primer orden, esta última puede expandirse a lógicas de ordenes superiores. En ellas las afirmaciones no solo pueden cuantificar sobre variables, sino también sobre las propios predicados y funciones. Esto permite una capacidad expresiva y un nivel de abstracción mucho mayor.

Ejemplo 2.22. Podríamos afirmar que $\forall x \forall y ((\forall F\, F(x) = F(y)) \rightarrow x = y)$, donde queremos simbolizar que si dos elementos del dominio se evalúan al mismo elemento para todas las funciones, entonces son en realidad iguales.

Otras fórmulas podrían ser $\exists x \forall P\, P(x)$ o $\forall P \exists x P(x)$, la primera simbolizando que existe un elemento que cumple todos los predicados y la segunda que para cada predicado existe un elemento que lo cumple.

Así como su sintaxis se agranda, también se expande su semántica. Podemos restringir los posibles dominios (semántica de Henkin) o tomar el conjunto potencia del conjunto de elementos. Con la primera se heredan las características de la lógica de primer orden, pero con la segunda el sistema deja de tener las buenas propiedades que hemos visto: *solidez* y *completitud*. Por ello si queremos aumentar la potencia expresiva del sistema, no nos queda otra que comprometer estas propiedades.

[3] Ver la Nota 1.

Capítulo 3

Teoría de tipos

Una *teoría de tipos* es la representación formal de un sistema de tipos. Estas pueden servir como alternativas para la fundamentación de las matemáticas constructivas o para el estudio de los formalismos en general. Muchos lenguajes de programación están basados en estas ideas para facilitar la corrección y validación del código. Sobre ellas, comentaremos brevemente su funcionamiento y las ideas fundamentales, ya que un análisis en profundidad queda fuera del alcance de este trabajo. En [5] se puede encontrar una descripción más detallada

En una teoría de tipos cada término es asignado a un tipo concreto, que define las operaciones posibles con otros términos. Tomamos la notación `término : tipo` para escribirlo. El ejemplo más famoso es el cálculo lambda tipado.

Ejemplo 3.1. Si consideramos el tipo $\mathbb{N}$ de los números naturales con sus operaciones habituales en notación prefija, los siguientes son términos:

- `5 :` $\mathbb{N}$
- `suma 4 1 :` $\mathbb{N}$
- `suma 1 (producto 2 2) :` $\mathbb{N}$

Notemos que si realizamos las operaciones (computación), los tres términos que en principio son distintos se reducen al mismo término. A partir de esta noción, se puede construir una relación de equivalencia entre términos.

Por otro lado, las funciones también son términos y como tales tienen su propio tipo. Los dos siguientes ejemplos representan la función suma y la función cuadrado en los naturales, para describirlas se utiliza la notación lambda:

- $(\lambda$`x:`$\mathbb{N}$`.(`λ`y:`$\mathbb{N}$`.(suma x y))) :` $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$
- $(\lambda$`x:`$\mathbb{N}$`.(producto x x)) :` $\mathbb{N} \to \mathbb{N}$

Asimismo, estas funciones se pueden aplicar a términos siempre que la sintaxis de sus tipos sea correcta. La primera expresión se reduce a la función que a 5 suma un natural y la segunda, al natural 9:

- $(\lambda$`x:`$\mathbb{N}$`.(`λ`y:`$\mathbb{N}$`.(suma x y))) 5 :` $\mathbb{N} \to \mathbb{N}$
- $(\lambda$`x:`$\mathbb{N}$`.(producto x x)) 3 :` $\mathbb{N}$

Recíprocamente si tenemos un término, podemos construir la función constante que devuelve ese término. Esta regla se conoce como λ*-abstracción*:

- $(\lambda$`x:`$\mathbb{N}$`.(5)) :` $\mathbb{N} \to \mathbb{N}$
- $(\lambda$`x:`$\mathbb{N}$`.(`λ`y:`$\mathbb{N}$`.(y))) :` $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$

En estos ejemplos el tipo parece siempre claro, pero no siempre es así. Pensemos ahora en el tipo de los *booleanos*, con `a` un término booleano y una función `si a b c` tal que `si verdadero b c` se reduce a `b` y `si false b c` se reduce a `c`.

Ahora nos hacemos la siguiente pregunta: ¿cuál es el tipo de `si a b c` en el caso de que `b` y `c` no sean del mismo tipo? Parece razonable añadir funciones que no solo devuelvan términos, si no tambien tipos. Por tanto, ha de haber algún tipo que contenga otros tipos; a los que llamamos *universos* y escribimos usualmente como `U`. Esto motiva la defincón de tipos dependientes que veremos en el siguiente apartado.

3.1. Tipos básicos

Las teorías de tipos están definidas por las reglas de inferencia aplicables a los términos de cada tipos. Acabamos de ver implícitamente en el Ejemplo 3.1 como se comportan las funciones, pero la mayoría de teorías definen también los siguientes tipos:

1. El tipo **vacío**.

   Usualmente escrito como $\bot$ o 0, no tiene ningún término y suele utilizarse para comprobar que algo no es computable. Por ejemplo si podemos crear un término de tipo $\texttt{A} \to \bot$, sabemos que el tipo `A` tampoco tiene términos.

2. El tipo **unidad**.

   Es el tipo con un único término canónico. El tipo se escribe $\top$ o 1 y el término es *. Se utiliza para comprobar la existencia, ya que si podemos construir un término de tipo $\top \to \texttt{A}$, sabemos que `A` tiene al menos un término.

3. El tipo **producto**.

   Es el tipo cuyos términos son pares ordenados. Si `A` y `B` son tipos, el producto es $\texttt{A} \times \texttt{B}$. Los términos son de la forma `par a b`, donde `a` y `b` son términos de tipo `A` y `B` respectivamente y `par` es la función constructora. Además se define junto a las funciones `primero` y `segundo` tales que:

   - `primero (par a b)` se reduce a `a`.
   - `segundo (par a b)` se reduce a `b`.

4. El tipo **suma**.

   El tipo suma es una "unión disjunta", así si `A` y `B` son tipos, la suma `A+B` tiene un término de tipo `A` o de tipo `B`, pero sabiendo de cual. Tiene dos constructores `inyIzquierda` e `inyDerecha`, de forma que `inyIzquierda a` e `inyDerecha b` son de tipo `A+B` si `a` y `b` son términos de tipo `A` y `B` respectivamente. Se define también la función `match` tal que si `C` es un tipo y tenemos $\texttt{f} : \texttt{A} \to \texttt{C}$ y $\texttt{g} : \texttt{B} \to \texttt{C}$:

   - `match (inyIzquierda a) C f g` se reduce a `f a`.
   - `match (inyDerecha b) C f g` se reduce a `g b`.

5. El tipo **producto dependiente**.

   Si tenemos una función en la que el tipo del valor devuelto depende del argumento, decimos que es una **función dependiente** y su tipo es el producto dependiente. Si `U` es un universo de tipos y $\texttt{A} : \texttt{U}$, tenemos una familia de tipos $\texttt{B} : \texttt{A} \to \texttt{U}$, que a cada término $\texttt{a} : \texttt{A}$ le asigna un tipo $\texttt{B(a)} : \texttt{U}$. El tipo producto dependiente $\prod_{\texttt{x}:\texttt{A}} \texttt{B(x)}$ está entonces formado por los términos que son funciones que toman un término $\texttt{a} : \texttt{A}$ y devuelven un término en `B(a)`.

   Notemos que en el caso de que `B(a)` sea constante, el tipo $\prod_{\texttt{x}:\texttt{A}} \texttt{B(x)}$ es esencialmente el tipo $\texttt{A} \to \texttt{B}$.

6. El tipo **suma dependiente**.

   De manera dual al anterior y siguiendo con el tipo `A:U` y la familia de tipos `B:A` $\to$ `U`, el tipo suma dependiente es $\sum_{x:A} B(x)$. Este tipo representa la idea de un par ordenado donde el tipo del segundo término depende del valor del primero, es decir, si `(a,b)` $: \sum_{x:A} B(x)$, entonces `a:A` y `B(a)`.

   Notemos de nuevo que si `B(a)` es constante, el tipo $\sum_{x:A} B(x)$ es esencialmente el tipo $A \times B$.

7. Tipos **inductivos**.

   Es un tipo en el que los términos son o constantes o el resultado de funciones con entradas del mismo tipo, de forma que cada término se obtiene de manera única. Estos tipos permiten (como su nombre indica) una recursión inductiva sobre ellos. El ejemplo más común son los números naturales construidos a partir de los Axiomas de Peano.

3.2. La correspondencia de Curry-Howard

Pese a que este sistema de tipos no tiene nada que ver en apariencia con los sistemas formales vistos en los capítulos primeros, ambos son en realidad equivalentes. En esencia, la correspondencia a la que se llega es que "*Una prueba es un programa y la fórmula que prueba es el tipo del programa*", entendiendo aquí que un programa es un término si lo miramos desde la perspectiva de su computación.

Primero Curry observó en 1934 que los tipos en el cálculo lambda se podían ver como axiomas o reglas en la lógica intuicionista y más tarde en 1969 Howard observó que la deducción natural se podía interpretar como una variante tipada del modelo de computación conocido como cálculo lambda. La idea principal es que las reglas de formación de términos y las reglas de inferencia para la derivación de fórmulas son equivalentes:

1. En la lógica, tenemos las proposiciones de la forma $A, B, \ldots$ y en la teoría de tipos, las variables libres `x:A`, `y:B`, $\ldots$ son términos.

2. La regla de derivación $\dfrac{\Gamma \vdash A \to B \qquad \Delta \vdash A}{\Gamma, \Delta \vdash B}$ (*Modus Ponens*) es equivalente a la aplicación de funciones $\dfrac{\texttt{f : A} \to \texttt{B} \qquad \texttt{g : A}}{\texttt{f g : B}}$.

3. La regla de introducción de la implicación $\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$ es equivalente a la λ-abstracción $\dfrac{\texttt{f : B}}{(\lambda \texttt{x:A.f}) \texttt{ : A} \to \texttt{B}}$.

Las ideas concretas de Curry y Howard sobre la relación entre la lógica proposicional y la teoría de tipos están explicadas más detalladamente en [7]. Mostramos en el Cuadro 3.1 las correspondencias que hay entre los dos sistemas. Para una justificación completa, ver [4, págs. 63-68] y [5, págs. 78-83].

Lógica formal	**Teoría de tipos**
Falsedad lógica	Tipo vacío
Verdad lógica	Tipo unidad
Implicación	Tipo función
Conjunción	Tipo producto
Disyunción	Tipo suma
Cuantificador universal	Tipo producto dependiente
Cuantificador existencial	Tipo suma dependiente
Sistema de la deducción natural	Teoría de tipos del cálculo lambda

Cuadro 3.1: Correspondencia de Curry-Howard

3.3. Asistentes de demostración: Lean

Los asistentes de demostración son herramientas capaces de garantizar que un razonamiento lógico es correcto. Utilizando las reglas de inferencia, axiomas e hipótesis, estos sistemas pueden verificar paso a paso la validez de una cadena de argumentos, asegurando que cada paso sea coherente con las reglas establecidas. Así se reduce la probabilidad de errores humanos e incluso se corrigen automáticamente si son sencillos, obteniendo del contexto los pasos necesarios.

Su aplicación abarca diversas áreas. En matemáticas, han permitido la demostración y exploración de teoremas complejos que podrían haber sido difíciles de verificar manualmente. En el ámbito de la ingeniería de software, estos asistentes desempeñan un papel crucial en la verificación formal de sistemas críticos, garantizando su correcto funcionamiento.

Actualmente existen varios proyectos independientes construidos sobre distintas tecnologías: *Lean*[1] sobre C++, *Isabelle* sobre Scala, *Mizar* sobre Pascal, *Coq* sobre OCaml, *Agda* sobre Haskell. Cada uno de ellos tiene además una gran librería que contiene la formalización y demostración de una gran cantidad de teoremas y conceptos de todos los campos de las matemáticas. Estas son mantenidas y desarrolladas libremente por sus comunidades y llegan a abarcar más allá del temario básico de un grado de matemáticas.

En este trabajo nos hemos decidido centrar en *Lean*, por ser uno de los que más funciones presenta y por su facilidad de uso. La librería correspondiente de *Lean* es `mathlib`. A fecha de 28 de agosto de 2023, el proyecto constaba de más de 1.128.000 líneas de código, 66.000 definiciones y 122.000 teoremas[2]. El Ejemplo 3.2, que veremos más adelante, es una construcción básica que se encuentra ya en `mathlib`.

Lean en esencia es un *comprobador de tipos* y por tanto equivalente a algún sistema lógico. Esto quiere decir que podemos escribir expresiones y preguntar al sistema si son correctas y a qué tipo de objeto denotan e interpretarlas como pruebas y fórmulas. Podemos definir variables de tipo proposición e hipótesis de cada expresión posible.

En *Lean 4*, la versión que utilizaremos a partir de aquí, la sintaxis es la siguiente:

```
variables A B C : Prop
variable h : A ∧ ¬ B → C
```

Formalmente *Lean* interpreta cada proposición como un tipo, el tipo de las pruebas de esa expresión. De esta forma, construir una prueba consiste en escribir expresiones del tipo correcto. Las reglas de inferencia funcionan como cabría esperar: si `P` es una expresión de tipo `A ∧ B`, entonces `and.left P` y `and.right P` son de tipo `A` y `B` respectivamente. Para aplicar Modus Ponens, basta con escribir las pruebas una después de otra, como si aplicáramos la primera a la segunda. Así si `P` es prueba de `A → B` y `Q` es prueba de `A`, entonces `P Q` es una prueba de `B`.

Para la introducción de la implicación, *Lean* utiliza la palabra reservada `assume`. Si con la hipótesis asumimos `A` y construimos `P`, una prueba de `B` a partir de ella, la expresión `assume h : A, P` es una prueba de `A → B`.

También podemos comunicar a *Lean* explícitamente nuestras intenciones con la palabra reservada `show`. Si `A` es una proposición y `P` es una prueba, `show A, from P` significa lo mismo que solamente `P`, pero indica a *Lean* que `P` ha de ser prueba de `A`.

```
example (A B : Prop) : A ∧ B → B ∧ A :=
assume h : A ∧ B,
show B ∧ A, from and.intro (and.right h) (and.left h)
```

Al construir un teorema o ejemplo, podemos usar las variables y premisas como argumentos y antes de la expresión o prueba debemos especificar a *Lean* el tipo.

[1] Sitio web del proyecto *Lean*: `https://leanprover.github.io/`

[2] Datos obtenidos de la página web de la comunidad: `https://leanprover-community.github.io/mathlib_stats.html`

Respecto a las igualdades, tras usar la palabra clave `by`, la táctica `rfl` demuestra objetivos del tipo `a = a` por la propiedad reflexiva. Si tenemos otro objetivo del tipo `a = b` y algunas hipótesis también de igualdades, podemos reescribir en la igualdad con las hipótesis hasta llegar a un objetivo del tipo `a = a` donde usar `rfl`. La táctica que lo permite es `rewrite` o `rw`. Lo ilustramos con el siguiente ejemplo:

```
variables a b c d : ℝ
variables h1 : b = c + d, h2 : a + c = d
example : a + b = d + d := by
    rw h1, rw h2, rfl
```

Equivalentemente se pueden poner las hipótesis en orden de utilización entre corchetes y al acabar *Lean* automáticamente comprueba si el objetivo se demuestra con `rfl`.

```
example : a + b = d + d := by rw [h1, h2]
```

Ejemplo 3.2. Para ilustrar el potencial de *Lean* vamos a construir los naturales mediante un tipo inductivo y demostrar propiedades de la suma y producto definidos sobre ellos.

```
inductive Naturales where
| cero : Naturales
| suc : Naturales → Naturales

def Naturales.suma : Naturales → Naturales → Naturales
  | a, cero   => a
  | a, Naturales.suc b => Naturales.suc (Naturales.suma a b)

def Naturales.mul : Naturales → Naturales → Naturales
  | _, cero   => cero
  | a, suc b => (Naturales.mul a b) + a
```

En la definición del tipo ponemos las posibles constantes y funciones constructoras, como en este caso es solo una (la función *sucesor*), esta es inyectiva.

Para definir las operaciones, lo hacemos recursivamente según los posibles casos, ya que cada término de tipo `Naturales` solo puede ser o `cero` o `suc a` para otro término `a:Naturales`.

Para demostrar la mayor parte de propiedades tendremos que usar la ventaja que nos proporciona el tipo inductivo: la *demostración por inducción*. Usando la táctica `induction`, *Lean* nos separa dos objetivos: demostrar el caso base (`cero`) y demostrar el paso de inducción para `suc n` con la hipótesis de inducción para `n`.

```
theorem suma_cero (a : Naturales) : a + cero = a := by rfl
theorem suma_suc (a b : Naturales) : a + suc b = suc (a + b) := by rfl

theorem cero_suma (a : Naturales) : cero + a = a := by
  induction a with
  | cero => rw [suma_cero] -- el objetivo es cero + cero = cero
  | suc a hi => rw [suma_suc, hi]
    -- el objetivo es cero + suc a = suc a; hi : cero + a = a
    -- usando rw suma_suc el objetivo pasa a suc (cero + a) = suc a
    -- usando rw hi el objetivo pasa a suc a = suc a
```

Siguiendo las mismas ideas, podemos demostrar las propiedades básicas: la asociatividad y conmutatividad de la suma y el producto y la distributividad de la suma respecto al producto. El código completo se encuentra en el Anexo.

Bibliografía

[1] E. MENDELSON, *Introduction to Mathematical Logic*, 6.ª ed., CRC Press, New York, USA, 2015.

[2] J. AVIGAD, R. Y. LEWIS Y F. VAN DOORN, *Logic and proof*, https://leanprover.github.io/logic_and_proof/.

[3] LaTeX FOR LOGICIANS, *bussproofs.sty A User Guide*, https://mathweb.ucsd.edu/~sbuss/ResearchWeb/bussproofs/BussGuide2_Smith2012.pdf.

[4] MORTEN HEINE B. SØRENSEN Y PAWEL URZYCZYN, *Lectures on the Curry-Howard Isomorphism*, Universidad de Copenhague & Universidad de Varsovia, 2006.

[5] SIMON THOMPSON, *Type Theory and Functional Programming.*, Universidad de Kent, 1999, https://www.cs.kent.ac.uk/people/staff/sjt/TTFP/ttfp.pdf.

[6] VÍCTOR LÓPEZ MARTÍNEZ, *El Teorema de Incompletitud de Gödel*, Universidad de Zaragoza, 2016, https://zaguan.unizar.es/record/59148.

[7] W. A. HOWARD, *The Formulae-As-Types Notion of Construction*, en *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, Academic Press, 1980.

Anexo: Código en Lean

Fichero *Naturales.lean* en Lean4

```
import Mathlib.Tactic.Basic
import Mathlib.Tactic.Cases
namespace Naturales

------------------------------------------------------------
-- Definicion inductiva de los numeros Naturales
inductive Naturales where
| cero : Naturales
| suc : Naturales → Naturales

-- Definicion recursiva de la suma
def Naturales.suma : Naturales → Naturales → Naturales
  | a, cero   => a
  | a, Naturales.suc b => Naturales.suc (Naturales.suma a b)

-- Para poder utilizar la notacion infija + con Naturales
instance : Add Naturales where
  add := Naturales.suma

-- Definicion recursiva de la multiplicacion
def Naturales.mul : Naturales → Naturales → Naturales
  | _, cero   => cero
  | a, suc b => (Naturales.mul a b) + a

-- Para poder utilizar la notacion infija * con Naturales
instance : Mul Naturales where
  mul := Naturales.mul

def uno : Naturales := Naturales.suc Naturales.cero

open Naturales
-- a partir de aqui las funciones definidas estan disponibles sin tener
    que acceder al paquete Naturales

------------------------------------------------------------
-- Teoremas sobre la suma en los numeros Naturales

-- Prueba de que el uno es el sucesor del cero
theorem uno_igual_suc_cero : uno = suc cero := by rfl

-- El cero es el elemento neutro de la suma a la derecha
theorem suma_cero (a : Naturales) : a + cero = a := by rfl

-- Por la definicion recursiva de la suma
theorem suma_suc (a b : Naturales) : a + suc b = suc (a + b) := by rfl
```

```
-- El cero es el elemento neutro de la suma a la izquierda
theorem cero_suma (a : Naturales) : cero + a = a := by
  induction a with
  | cero => rw [suma_cero] -- el objetivo es cero + cero = cero
  | suc a hi => rw [suma_suc, hi]
    -- el objetivo es cero + suc a = suc a; hi : cero + a = a
    -- usando rw suma_suc el objetivo pasa a suc (cero + a) = suc a
    -- usando rw hi el objetivo pasa a suc a = suc a

lemma suma_asociativa (a b c : Naturales) : (a + b) + c = a + (b + c) := by
  induction c with
  | cero => rw [suma_cero, suma_cero]
  | suc c hi => rw [suma_suc, suma_suc, suma_suc, hi]

theorem suc_suma (a b : Naturales) : suc a + b = suc (a + b) := by
  induction b with
  | cero => rw [suma_cero, suma_cero]
  | suc b hi => rw [suma_suc, suma_suc, hi]

lemma suma_conmutativa (a b : Naturales) : a + b = b + a := by
  induction b with
  | cero => rw [suma_cero, cero_suma]
  | suc b hi => rw [suma_suc, suc_suma, hi]

-- El sucesor es la suma del numero con el uno
theorem suc_igual_suma_uno (a : Naturales) : suc a = a + uno := by
  rw [uno_igual_suc_cero, suma_suc, suma_cero]

------------------------------------------------------------
-- Teoremas sobre la multiplicacion en los únmeros Naturales
-- El cero es el elemento absorbente de la multiplicacion a la derecha
theorem mul_cero (a : Naturales) : a * cero = cero := by rfl
-- Por la ódefinicin recursiva de la ómultiplicacin
theorem mul_suc (a b : Naturales) : a * suc b = a * b + a := by rfl

-- El cero es el elemento absorbente de la multiplicacion a la izquierda
theorem cero_mul (a : Naturales) : cero * a = cero := by
  induction a with
  | cero => rw [mul_cero]
  | suc a hi => rw [mul_suc, suma_cero, hi]

-- El uno es el elemento neutro de la multiplicacion a la derecha
theorem mul_uno (a : Naturales) : a * uno = a := by
  rw [uno_igual_suc_cero, mul_suc, mul_cero, cero_suma]

-- El uno es el elemento neutro de la multiplicacion a la izquierda
theorem uno_mul (a : Naturales) : uno * a = a := by
  induction a with
  | cero => rw [mul_cero]
  | suc a hi => rw [mul_suc, hi, uno_igual_suc_cero, suma_suc, suma_cero]

-- La multiplicacion es distributiva a la izquierda con respecto a la suma
theorem distributiva_izda (a b c : Naturales) : a * (b + c) = a * b + a *
    c := by
  induction c with
  | cero => rw [suma_cero, mul_cero, suma_cero]
  | suc c hi => rw [suma_suc, mul_suc, hi, mul_suc, suma_asociativa]
```

```
lemma mul_asociativa (a b c : Naturales) : (a * b) * c = a * (b * c) := by
  induction c with
  | cero => rw [mul_cero, mul_cero, mul_cero]
  | suc c hi => rw [mul_suc, mul_suc, hi, distributiva_izda]

theorem suc_mul (a b : Naturales) : suc a * b = a * b + b := by
  induction b with
  | cero => rw [mul_cero, mul_cero, suma_cero]
  | suc b hi => rw [mul_suc, mul_suc, hi, suma_asociativa, suma_asociativa,
      suma_suc b a, suma_suc a b, suma_conmutativa a b]

-- La multiplicacion es distributiva a la derecha con respecto a la suma
theorem distributiva_dcha (a b c : Naturales) : (a + b) * c = a * c + b *
    c := by
  induction b with
  | cero => rw [cero_mul, suma_cero, suma_cero]
  | suc b hi => rw [suma_suc, suc_mul, hi, suc_mul, suma_asociativa]

theorem mul_conmutativa (a b : Naturales) : a * b = b * a := by
  induction b with
  | cero => rw [mul_cero, cero_mul]
  | suc b hi => rw [mul_suc, suc_mul, hi]
```

Índice alfabético