Trabajo de Fin de Máster

Teoría de tipos con funciones parciales

Joaquín Castro Abal
Supervisado por: Víctor Aranda Utrero
Universidad de Valladolid
1 de julio de 2024

Resumen: En este trabajo se estudia la teoría de tipos con funciones parciales. Para llevar a cabo este estudio, se completa un amplio recorrido. El trabajo parte del contexto en el que nace la teoría de tipos y con qué propositos. Seguidamente se exponen los sistemas de teoría de tipos $\mathcal{Q}_0$, Alonzo, PF y LUTINS, de los cuales los tres últimos incluyen funciones parciales. La presentación de los fundamentos de cada sistema permite compararlos y analizar su adecuación para la formalización de las matemáticas y la implementación en computación. Finalmente, se concluye que la teoría de tipos, especialmente con funciones parciales, presenta múltiples ventajas en estos ámbitos. Por este motivo, se presenta también una propuesta pedagógica para la incorporación de la teoría de tipos en la educación general y en las carreras ligadas a las matemáticas aplicadas y computación.
Palabras clave: formalización de las matemáticas, términos no denotativos, semántica, regla de inferencia, computación, pedagogía.

Abstract: In this project type theory with partial functions is studied. To carry out this investigation, an extensive journey is completed. The work starts from the context in which type theory was born and for what purposes. Next, the type theory systems $\mathcal{Q}_0$, Alonzo, PF and LUTINS are presented, the last three including partial functions. The presentation of the fundamentals of each system allows to compare them and analyze their suitability for the formalization of mathematics and implementation in computing. Finally, it is concluded that type theory, especially with partial functions, has multiple advantages in these areas. For this reason, a pedagogical proposal is also presented for the incorporation of type theory to general education and careers linked to applied mathematics and computing.
***Keywords*:** Formalization of mathematics, non-denoting terms, semantics, rule of inference, computing, pedagogy.

Índice

1. Introducción

1.1. Objetivos

El objetivo principal de este trabajo es investigar las virtudes de la teoría de tipos con funciones parciales. Para ello, se pretende estudiar la génesis de la teoría de tipos y su evolución. Además, otro de los enfoques del trabajo es comprender los formalismos de algunos sistemas de teoría de tipos con funciones totales y otros con funciones parciales. Se pretende evaluar la adecuación de estos sistemas para la formalización de las matemáticas y la computación, con especial atención a la hipótesis de que la teoría de tipos con funciones parciales es la más óptima debido a su naturalidad[1]. Además, se plantea la incorporación de estas lógicas en el sistema educativo, particularmente en las carreras STEM[2].

Para lograr los objetivos generales, se establecen los algunos objetivos específicos. En primer lugar, se explorará el contexto histórico de la teoría de tipos, identificando los principales hitos y contribuciones que han marcado su desarrollo. Este objetivo busca proporcionar una visión integral del contexto histórico y los avances que han influido en la formulación y el perfeccionamiento de la teoría de tipos. En segundo lugar, se realizará una presentación formal y detallada de los sistemas de teoría de tipos $\mathcal{Q}_0$, Alonzo, PF y LUTINS, destacando sus estructuras, principios semánticos y fundamentos. Este análisis permitirá comprender en profundidad cómo cada uno de estos sistemas puede ser capaz de abordar la formalización de conceptos matemáticos y problemas computacionales. En tercer lugar, se evaluará la adecuación de los sistemas presentados para la formalización de las matemáticas y la computación, poniendo un énfasis especial en la teoría de tipos con funciones parciales. La hipótesis central es que este enfoque ofrece un razonamiento matemático acorde a la práctica matemática, lo que podría traducirse en una mayor eficiencia para la implementación informática de softwares dedicados a la verificación de pruebas. Finalmente, se planteará y analizará la hipótesis de la integración óptima de los sistemas de teoría de tipos en la educación STEM, investigando cómo la inclusión de la lógica tipos en etapas tempranas de la educación podría beneficiar la formación de los estudiantes. Este objetivo también contempla la aplicación práctica de los sistemas de teoría de tipos con funciones parciales en programas específicos dentro de las carreras STEM.

[1] Se entiende naturalidad como el razonamiento habitual en la práctica matemática. En las matemáticas, por ejemplo, se asume que las funciones tienen un dominio de aplicación concreto, el cual no siempre coincide con la totalidad de elementos del universo.

[2] STEM es el acrónimo del inglés *Science, Technology, Engineering and Mathematics.*

1.2. Contexto

La teoría de tipos es un sistema formal que sirve como alternativa a la teoría de conjuntos como fundamento de las matemáticas, así como herramienta para el diseño, análisis y estudio de los sistemas de tipos en los lenguajes de programación. La teoría de tipos fue un pilar esencial en la estructura del pensamiento lógico y matemático del siglo XX y surgió como respuesta a los problemas y paradojas que afectaban a la teoría de conjuntos original de Cantor y Frege, quienes pretendían formalizar el concepto intuitivo de colección de objetos.

El proyecto de Frege comenzó con la publicación de *Begriffsschrift*, cuya obra contiene la primera versión de su sistema lógico [1]. Uno de los principales interesados en las obras de Frege fue Bertrand Russell. Russell trabajó también en la fundamentación de las matemáticas a través de la lógica. De hecho, durante unos años Russell y Frege se cartearon para tratar determinadas cuestiones relacionadas con ests temáticas. En una primera carta, Russell comunicó a Frege su interés en la obra *Grundgesetze der Arithmetik*, cuyo segundo volumen sería publicado un año después, en 1903 [2]. En esa misma carta, Russell transmite a Frege que ha hallado una problemática en el *Begriffsschrift*. Esta problemática se relaciona con el esquema de comprensión de Frege y tiene un enfoque desde la lógica y la teoría de conjuntos, poniendo especial atención en cómo Frege define las funciones. Así pues, Rusell comunica a Frege:

> I have encountered a difficulty only on one point. You assert (p.17) that a function could also constitute the indefinite element. This is what I used to believe, but this view now seems to me dubious because of the following contradiction: Let w be the predicate of being a predicate which cannot be predicated of itself. Can w be predicated of itself? From either answer follows its contradictory. We must therefore conclude that w is not a predicate. Likewise, there is no class (as a whole) of those classes which, as wholes, are not members of themselves. From this I conclude that under certain circumstances a definable set does not form a whole. (Russell, 1902, p.130-131) [2].

Russell dió con esta antonimia en las obras de Frege mientras trataba de descubrir alguna falla en la prueba de Cantor de que no existe un cardinal mayor que todos los demás. Por lo tanto, el teorema de Cantor jugó un papel relevante también en toda esta historia [3]. El teorema en cuestión, nos dice que no existe un mapeo sobreyectivo de un conjunto a su conjunto potencia; esto es, $f : X \to \wp(X)$, es decir, intuitivamente un conjunto tiene más subconjuntos que elementos. Consideremos el siguiente subconjunto de X: $A = \{x \in X \mid x \notin F(x)\}$. Este subconjunto no puede estar en el rango de f . Porque si $A = F(a)$, para algún a , entonces $a \in F(a)$ syss $a \in A$ syss $a \notin F(a)$, lo cual es una contradicción. Por otra parte, Russell analizó qué sucede al considerar que A es la clase de todas las clases y más delante forzó la consideración de la clase

de todas las clases que no pertenecen a sí mismas, es decir, $R = \{w \mid w \notin w\}$. En ese caso, al evaluar qué ocurre con la clase en sí misma tenemos que $R \in R$ si y solo si $R \notin R$, lo cual es contradictorio. De hecho, Cantor ya era consciente del hecho de que la clase de todos los conjuntos no puede considerarse en sí misma como un conjunto[3].

Por otra parte, Frege trató de aportar una solución a la paradoja, aunque en el sistema que describió la paradoja se expresa de forma diferente. Frege escribió que la expresión "un predicado se predica por sí mismo" no es exacta así que hizo una distinción entre predicados (conceptos) y objetos. Un predicado (de primer orden) se aplica a un objeto pero no puede tener un predicado como argumento. La formulación exacta de la paradoja en el sistema de Frege utiliza la noción de extensión de un predicado P, que designamos como εP. La extensión de un predicado es en sí misma un objeto. El que lo describe es: $\varepsilon P = \varepsilon Q \equiv \forall x[P(x) \equiv Q(x)]$. Así pues, la paradoja se extiende como $R(x)$ syss $\exists P[x = \varepsilon P \wedge \neg P(x)]$, usando el axioma anterior, $R(\varepsilon R) \equiv \neg R(\varepsilon R)$ [3]. No obstante, la solución que más luces aportó a la lógica fue la que propuso Russell: la teoría de tipos. La teoría de tipos evita la paradoja estableciendo una jerarquía en la que el predicado de pertenencia discrimina entre tipos lógicos. Por lo tanto, el universo en teoría de tipos no es plano, lo que provoca que no se pueda formular la paradoja, ya que sintácticamente no tiene sentido [4].

Posteriormente, el interés en resolver la paradoja se convirtió en algo general. De hecho, los lógicos de la época no estaban tan solo interesados en la paradoja, sinó también en la formalzación lógica de las matemáticas. Como ya hemos visto, Frege fue uno de los implicados y Russell fue el encargado de colocar el primer ladrillo en la construcción de la teoría de tipos. Russell trabajó también con Whitehead estudiando los fundamentos de las matemáticas. De hecho, esta colaboración culminó en la brillante obra *Principia Mathematica*. Paralelamente, hubo muchos otros que aportaron en la formulación detallada de la teoría de tipos simple con sus propias versiones, entre ellos Zermelo, Quine, o Neumann. Dichos autores tuvieron todo un enfoque en la teoría de conjuntos pero Quine y Neumann, a diferencia de Zermelo, siguieron una estrategia más radical: alterar la lógica subyacente con una aproximación paraconsistente [5].

En definitiva, podemos darnos cuenta de que fueron muchos los autores que surgieron en esta época. Todos trabajando en sus campos e inquietudes concretas pero siempre con un denominador común. De hecho, este periodo tiene nombre propio y ha pasado a la historia. La crisis de las matemáticas del siglo XX, también conocida como la crisis de los fundamentos, fue un período de reflexión profunda y debate sobre los cimientos y la dirección de la disciplina matemática. Este período se caracterizó por varios eventos y desarrollos clave.

[3]Cuando esta paradoja se exitiende de forma que la clase definible corresponde a las proposiciones recibe el nombre de *Paradoja de Russell-Myhill*.

En primer lugar, ya hemos visto que aparecieron algunas fuertes paradojas en la teoría de conjuntos, como la paradoja de Russell, que cuestionaron la consistencia de los fundamentos matemáticos. En segundo lugar, David Hilbert propuso un ambicioso programa para establecer una base sólida para todas las matemáticas, basada en un sistema completo y consistente de axiomas [6]. Por otra parte, Kurt Gödel aportó grandes avances a la lógica y las matemáticas con su estudio a la hipótesis del contínuo y los teoremas de incompletitud. Tal fue la magnitud de sus investigaciones que Gödel demostró en 1931 que cualquier sistema suficientemente poderoso para incluir la aritmética no puede ser al mismo tiempo completo y consistente. Esto significaba que siempre habría afirmaciones verdaderas en matemáticas que no podrían ser probadas dentro del sistema [7]. Además, gracias a los números de Gödel se consiguió una aritmética autorreferencial [8], lo cual supuso un gran avance también para la computación, una disciplina íntimamente ligada a la teoría de tipos que nos concierne. No obstante, este hallazgo derivó también en un temor a la aparición de paradojas como la del mentiroso dentro de la aritmética. Así pues, a pesar del nombre que se le adjudicó a ese periodo, la crisis de los fundamentos aumentó el rigor matemático y la comprensión de que las matemáticas no son infalibles, sino que están sujetas a limitaciones inherentes a su estructura lógica. Por lo tanto, este período fue crucial para el desarrollo de la lógica matemática y la filosofía de las matemáticas, y sus repercusiones se sienten hasta el día de hoy.

Una vez consolidada la lógica como una disciplina, aparece la figura de Alonzo Church, uno de los mayores protagonistas de este trabajo. En la década de 1940, Alonzo Church no pasó desapercibido entre todos los grandes lógicos de la época. Esto fue gracias a su aportación a la teoría de tipos. Comúnmente, Church es conocido como el padre de la teoría de tipos. Esto fue por su modificación a la teoría de tipos de Russell, introduciendo la notación λ. Este refinado de la teoría de tipos incorpora la abstracción λ y la definición por recursión primitiva de una forma muy elegante, por lo tanto, mejora algunas versiones de los precursores a los que el mismo Church hace referencia en su artículo: "we are heavily indebted to Whitehead, Russell, Hilbert, Ackermann Bernays" (A.Church, 1940, p.1) [9]. Este sistema propio de Church daría pie al conocido cálculo λ. María Manzano bautizó al cálculo λ como la piedra filosofal de Church por sus aportaciones al problema de la decisión y a la recursividad en computación [10]. El impacto de la teoría de tipos fue en aumento con el paso del tiempo ya que en la primera versión de la teoría de tipos no se pudo demostrar la completitud.

En esta etapa, existía una cierta confusión entre los términos completud y decibilidad. Aunque muchos autores perseguían una lógica decidible o computable, consiguieron sistemas lógicos completos. De hecho, Henkin no trataba de demostrar sus teoremas de completitud cuando los

demostró. Sin embargo, las pruebas de completud y la formalización lógica de las matemáticas con la teoría de tipos como herramienta fueron piezas clave. Asimismo, fue una conjunción de factores los que hicieron posible que Henkin desarrollara en la década de 1950 su propio sistema de teoría de tipos con una modificación en la semántica de Church, la cual ya incluía la notación λ [11]. Esta modificación colocó a Church en lo más alto ya que gracias a esta versión, la teoría de tipos de Church (notación λ) sería completa. La forma en la que Henkin consiguió esta prueba de completud para la teoría de tipos con notación λ fue introduciendo cambios en la semántica. La semántica estándar no permite mantener el poder de computabilidad sin sacrificar la capacidad expresiva del lenguaje. Henkin se dió cuenta de que en la formulación de la semántica de la SOL[4] se cometieron algunas asunciones quizás injustificadas, pues la metateoría de conjuntos de Zermelo Frenkel estaba demasiado ligada al lenguaje objeto. De hecho, la noción de subconjunto se convierte en un concepto lógico más, es así como nace la semántica no estándar [13].

En la década siguiente, la teoría de tipos siguió creciendo y se mantuvo mucho el interés en la disciplina. No obstante, en los años posteriores el interés general en la teoría se fue difuminando, aunque no se abandonó por completo. De hecho, en la década de los 90 se reavivó la cuestión gracias a los avances en computación, los cuales a su vez aportan mucho beneficio económico. Por ese motivo, actualmente la teoría de tipos está más ligada a la computación que a la formalización de las matemáticas. De hecho, han sido varios los sistemas de prueba de teoremas de orden superior los que se han implementado. Además también se han desarrollado lenguages de programación relacionados con la temática. Algunos ejemplos son LISP, IMPS [11] o LEO-II y LEO-III [12].

Atendiendo a la influencia en el ámbito computacional, la presencia de Alan Turing, cuya tesis fue dirigida por Church, fue también fundamental. Después de que Kurt Gödel demostrara sus teoremas de incompletitud, Turing avanzó en el campo de la lógica y la computación con su propio trabajo seminal: los números computables y una aplicación al problema de la decisión. El modelo teórico de una máquina que puede realizar cálculos se convirtió en la base de lo que hoy conocemos como computadora y sirvió también como respuesta al problema de decisión de Hilbert a partir de determinar si existía un algoritmo que pudiera determinar la veracidad o falsedad de cualquier afirmación matemática. Turing demostró que no existe tal algoritmo universal, lo que significa que hay problemas que no pueden ser resueltos por un procedimiento mecánico, estableciendo así los límites de la calculabilidad [14]. Más adelante, mostraría también la equivalencia entre las definiciones de Gödel y Church de la computabilidad efectiva. Para Gödel, la computabilidad efectiva correspondía a las funciones definidas bajo recursión y para Church a las funciones definibles con el operador λ [15]. Estas definiciones, al fin y al cabo, recitan

[4]Abreviatura de "Second Order Logic".

con otras palabras lo que es una función Turing-computable con una máquina abstracta. En definitiva, el trabajo de Turing no solo tuvo implicaciones profundas para la lógica y las matemáticas, sino que también sentó las bases para el la génesis de la informática y la inteligencia artificial.

Dicho todo esto, es posible realizar un breve análisis del contexto que atienda a las cuestiones que más interesan para el trabajo: la teoría en sí misma, los métodos pedagógicos posibles y la relación con las distintas disciplinas.

Adentrándonos en un contexto más teórico de la cuestión, en teoría de conjuntos, un tipo se interpreta simplemente como un conjunto, y los tipos de funciones $A \to B$ se pueden explicar utilizando la noción de teoría de conjuntos de función (como una relación funcional, es decir, un conjunto de pares de elementos). El tipo $A \to \circ$ corresponde a la operación de conjunto potencia, donde $\circ$ es el tipo de los valores de verdad.

En primer lugar, en la teoría de conjuntos, un conjunto es definible si existe una relación n-aria de ese conjunto tal que los objetos que lo forman satisfacen una propiedad específica que puede ser expresada mediante una fórmula en el lenguaje formal de la teoría en la cual se está trabajando. Esta fórmula debe distinguir precisamente a los elementos del conjunto en cuestión dentro del universo de discurso, es decir, debe definir inequívocamente los elementos que pertenecen al conjunto. En segundo lugar, el axioma de comprensión es fundamental ya que sirve para formalizar la idea de que se pueden formar subconjuntos a partir de un conjunto existente, siempre que se cumpla una condición específica. Este axioma supone que cada fórmula abierta y bien formada[5] determina un concepto cuya extensión existe y es el conjunto de todos aquellos elementos que satisfacen el concepto. El axioma de comprensión se formula de la siguiente manera para cada fórmula abierta φ: $\exists y \forall x[(x \in y) \Leftrightarrow \varphi(x)]$, donde y no es una variable libre en $\varphi(x)$. Una comprensión tan ilimitada conduce a la siguiente paradoja: tomamos $\varphi(x)$ como $\neg(x \in x)$, entonces, $\exists y \forall x[(x \in y) \Leftrightarrow \neg(x \in x)] \Rightarrow \exists y[(y \in y) \Leftrightarrow \neg(y \in y)]$. Esta paradoja se evita modificando los axiomas. Por ejemplo, el axioma de comprensión se modifica para obtener el axioma de separación: $\exists y \forall x[(x \in y) \Leftrightarrow (x \in z) \wedge \varphi]$, donde y no tiene apariciones en φ.

El enfoque para evitar la paradoja en teoría de tipos es completamente diferente del enfoque en teoría de conjuntos. En primer lugar, la solución de Zermelo modifica los axiomas de la teoría de conjuntos mientras mantiene un lenguaje lógico estándar, mientras que Russell modifica el propio lenguaje lógico. Es decir, en teoría de tipos es el lenguaje el que se altera para evitar la paradoja, y no los axiomas. Además, desde que se extendió el cálculo λ de Church con tipos

[5] A menudo aparecerá fbf para hacer referencia a una fórmula bien formada.

simples, la teoría de tipos ha seguido centrándose en la noción de función en lógica y matemáticas, pues las funciones han seguido siendo uno de los principales objetos de estudio en teoría de tipos.

En cuanto al análisis pedagógico, la teoría de tipos fue una disciplina meramente vista como objeto de investigación. No sería hasta más adelante que se empezaría a enseñar la teoría de tipos en cursos de lógica matemática y teoría de la computación en universidades, especialmente en departamentos de matemáticas y ciencias de la computación. Posteriormente, de la década de 1990 en adelante, con el crecimiento de la programación funcional y la importancia de los sistemas de tipos en la teoría de la computación, los cursos específicos sobre teoría de tipos se hicieron más comunes en los programas de ciencias de la computación. Esto demuestra la posibilidad de enseñar teoría de tipos aplicada a ciertos campos sin una base sólida de toda la lógica que hay detrás. Por lo tanto, la didáctica de la teoría de tipos puede empezar por los sistemas más aplicados para así poder luego entender las notaciones más complejas y sus fundamentos.

Finalmente, a todo esto podemos añadir una pequeña reflexión sobre la relación de la lógica con las matemáticas. Aunque podemos estar de acuerdo en que ambas disciplinas están intrínsecamente relacionadas, su interacción y percepción mútua han evolucionado con el tiempo. Durante la primera mitad del s.XX, la lógica y las matemáticas a menudo se consideraban disciplinas separadas. La lógica, aunque fundamental para el desarrollo de la teoría de conjuntos y la axiomatización de las matemáticas, no siempre se percibía como parte integral de las matemáticas aplicadas. La presencia de figuras como Gödel, Turing o Church contribuyó a la importancia de la lógica para las ciencias de la computación y las matemáticas, lo que convirtió a la lógica en una rama importante de las matemáticas, cambiando el tipo de preguntas que podrían ser respondidas por los métodos matemáticos y revelando sus limitaciones [6].

2. Marco teórico

En las próximas secciones, comentaremos las algunas aspectos generales de la teoría de tipos; así como las principales características sintácticas y semánticas de los sistemas de teoría de tipos que trataremos: $\mathcal{Q}_0$, Alonzo, PF y LUTINS. Las tres últimas versiones incluyen funciones parciales.

2.1. Teoría de tipos de Church

Más allá de la lógica clásica de primer orden, la SOL se consigue, en líneas generales, mediante la cuantificación sobre relaciones $n-$arias, añadiendo al lenguaje variables relacionales. Similarmente, la lógica de tercer orden se consigue introduciendo variables de predicados y funciones de tipo superior para denotar relaciones y funciones cuyos argumentos pueden ser relaciones y funciones de individuos así como individuos. Mediante este procedimiento pueden conseguirse sucesivamente lógicas de órden superior.

Evidentemente, al aumentar considerablemente el orden de estas lógicas uno se queda sin letras para denotar varibales. Denotaremos con $\mathcal{F}^\omega$ el sistema de orden ω que contiene todos los subsistemas de orden finitos que, por simplicidad, no tiene símbolos de función. Esto es posible ya que toda función n-aria puede ser expresada como una relación $(n+1)$-aria. Bien, este sistema es también conocido como la teoría finita de tipos y puede servir como una buena introducción a la disciplina. Más adelante se introducirá una formulación más óptima para este sistema denotado por $\mathcal{Q}_0$; este es, el sistema simplificado de Andrews para la teoría de tipos de Church.

2.1.1. Sistema $\mathcal{Q}_0$

Al igual que las fórmulas del lenguaje, los símbolos también se pueden definir recursivamente:
$\imath$ de orden 0. Denota los tipos de los individuos.
$\circ$ de orden 1. Denota el tipo de los valores de verdad y se adjudica a variables proposicionales.
Si $\tau_1, ..., \tau_n$ son símbolos de tipo, entonces $(\tau_1 ... \tau_n)$ es un símbolo de tipo que denota el tipo de las relaciones n-arias con argumentos de tipo $\tau_1, ..., \tau_n$ respectivamente y su orden es 1+(el máximo de los órdenes de $\tau_1, ..., \tau_n$).

Por otra parte, los símbolos primitivos de $\mathcal{F}$ son los habituales: $\neg, \wedge, \vee, \exists, \forall$ y los auxiliares $(,)$. Además de una lista numerable de variables de cada tipo y, opcionalmente, constantes de algunos tipos. Con todo esto se pueden definir ahora las reglas de formación para $\mathcal{F}^\omega$:
Cada variable proposicional es una fórmula bien formada.
Si $u_{\tau_1 ... \tau_n}, v^1_{\tau_1} ..., v^n_{\tau_n}$ son constantes o variables del tipo indicado por el subíndice, entonces

$u_{\tau_1 \ldots \tau_n} v^1_{\tau_1} \ldots v^n_{\tau_n}$ es una fbf.[6]
Si A y B son fbfs y u es una variable de tipo cualquiera, entonces $\neg A, A \vee B, A \wedge B, \exists u A$ y $\forall u A$ son también fbfs.
El orden de cada variable o constante lo denota el orden de su tipo.
La igualdad se puede definir de la siguiente manera: Si x_τ y y_τ son variables o constantes del mismo tipo, definimos $x_\tau = y_\tau$ como $\forall z_{(\tau)}[z_{(\tau)} x_\tau \supset z_{(\tau)} y_\tau]$, donde $z_{(\tau)}$ es una variable de tipo (τ). Cabe resaltar que todas las propiedades de la igualdad se siguen de la reflexividad y la sustituibilidad en una forma suficientemente fuerte.

El esquema de axiomas para $\mathcal{F}^\omega$ es [16]:

1. $(A \vee A) \supset A$.
2. $A \supset (B \vee A)$.
3. $(A \supset B) \supset ((C \vee A) \supset (B \vee C))$.
4. $\forall x_\tau A \supset \mathrm{S}^{x_\tau}_{y_\tau} A$ donde x_τ e y_τ son del mismo tipo, siendo y_τ una variable libre.
5. $\forall (A \vee B) \supset (A \vee \forall x B)$ donde x es cualquier variable no libre en A.
6. Axiomas de comprensión: $\exists u_o (u_o \equiv A)$, donde u_o no tiene apariciones libres en A.
   $\exists u_{(\tau_1 \ldots \tau_n)} \forall v^1_{\tau_1} \ldots \forall v^n_{\tau_n} (u_{(\tau_1 \ldots \tau_n)} v^1_{\tau_1} \ldots v^n_{\tau_n} \equiv A)$, donde u_o no es libre y $v^1_{\tau_1}, \ldots, v^n_{\tau_n}$ son distintas.
7. Axiomas de extensionalidad: $(x_o \equiv y_o) \supset (x_o = y_o)$.
   $\forall w^1_{\tau_1} \ldots \forall w^n_{\tau_n} (x_{\tau_1 \ldots \tau_n} w^1_{\tau_1} \ldots w^n_{\tau_n} \equiv y_{(\tau_1 \ldots \tau_n)} w^1_{\tau_1} \ldots w^n_{\tau_n}) \supset (x_{(\tau_1 \ldots \tau_n)} = y_{(\tau_1 \ldots \tau_n)})$.

Los axiomas de comprensión aparecen en segundo orden y los de extensionalidad en cuarto orden. Por lo tanto, juntamente con los axiomas 6 y 7, mediante la adición de variables de orden arbitrario y cuantificación sobre ellas se obtienen lógicas de orden superior. Adicionalmente, hay otros axiomas que pueden ser considerados, entre ellos se hallan los axiomas de elección, el axioma del infinito e incluso la hipótesis del continuo. Es importante decir también que se puede amenizar el lenguaje de forma que todos los predicados sean monádicos. De esta manera, escribimos $v_\tau \in u_\tau$ en vez de $u_\tau v_\tau$ y decimos "v_τ está en u_τ" en vez de "v_τ tiene la propiedad u_τ".

En teoría de conjuntos, $\in$ es una relación binaria y todas las fbfs son de la forma $x \in y$. Así, el axioma de comprensión queda como $\exists u \forall v (v \in u \equiv A)$ donde u no tiene apariciones libres. Este axioma afirma que para cada fbf A en la que v ocurre libre, hay un conjunto u que consiste en todos esos elementos v en los que A es verdadera. Desafortunadamente, esto garantiza la existencia de demasiados conjuntos, en especial, $\exists u \forall v (v \in u \equiv \neg v \in u)$; esto es, la paradoja de Russell. Uno

[6] Habitualmente se usará fbf para hacer referencia a una fórmula bien formada.

podría tomar cuidadosamente algunas instancias de los axiomas de comprensión y tomarlas como axiomas para obtener una teoría de conjuntos aparentemente consistente. Alternativamente, uno puede mantener los símbolos de tipos y evitar la paradoja de Russell de tal manera que no pueda ser ni siquiera formulada en ese sistema formal.

Por otra parte, la teoría de tipos es intuitivamente una forma más natural de formalizar matemáticas ya que en cualquier caso se emplean distintos tipos de letras para denotar distintos tipos de objetos matemáticos. Por lo tanto, la teoría de tipos se centra en hallar una formulación lo más expresiva posible que permita expresar ideas matemáticas con precisión, de la forma más simple y evitando ambigüedades. Además, presenta grandes utilidades.

Si queremos mantener las funciones en el lenguaje dada su gran utilidad práctica, conviene hablar del tipo que les corresponde. Una función de tipo $(\alpha\beta)$ representa una función de elementos de tipo β a elementos de tipo α[7]. En general, a una función que mapea n-tuplas $\langle x^1_{\beta_1}, ..., x^n_{\beta_n}\rangle$ de elementos de tipo $\beta_1...\beta_n$ a elementos de tipo α, se le asigna el tipo $((...(\alpha\beta_n)...\beta_2)\beta_1)$, simplificando, $(\alpha\beta_n...\beta_1)$. Un conjunto (o una propiedad) se puede representar con una función que mapea elementos a valores de verdad. Asimismo, un elemento pertenecerá al conjunto si y solo la función que representa al conjunto mapea al elemento con el valor verdadero. Siendo $\circ$ el tipo de los valores de verdad, hablaremos de una función de tipo $(\circ\alpha)$ para referirnos a un conjunto con elementos de tipo α. Así, $((\circ\alpha)\beta)$ es una relación binaria entre elementos de tipo β y elementos de tipo α. Por ejemplo, sea σ el tipo de los naturales y $<$ la relación de orden, el tipo de dicha relación es $(\circ\sigma\sigma)$, pues escribimos $<xy$, más comúnmente $x < y$, siendo x e y naturales. Uno puede darse cuenta del comportamiento de los tipos. Los tipos funcionan de forma similar a los conjuntos pero no son exactamente conjuntos ya que los tipos proporcionan también información sintáctica adicional.

En la teoría de tipos de Church aparece un nuevo símbolo que es, hasta cierto punto, similar a los cuantificadores $\exists$ y $\forall$ aunque con un significado distinto. Este es el símbolo λ. $(\lambda v_\beta A_\beta)$ denota la función cuyo valor en cualquier argumento v_β es A_α, donde v_β debe tener apariciones libres en A_α. Por ejemplo, $(\lambda n_\sigma(n^2_\sigma + 3))$ donde la función cuyo valor en cualquier natural n es $n^2 + 3$, es decir, $(\lambda n_\sigma(n^2_\sigma + 3))5 = 5^2 + 3 = 28$. Asímismo, en términos de λ, el axioma de comprensión se convierte en $\forall v_\beta((\lambda v_\beta A_\alpha)v_\beta) = A_\alpha$. Al proceso de reemplazar $((\lambda v_\beta A_\alpha)B_\beta$ por $S^{v_\beta}_{B_\beta}A_\alpha$ se le llama conversión λ. Por lo tanto, si tenemos $\alpha = 0$ se denota $((\lambda v_\beta A_0)B_\beta = S^{v_\beta}_{B_\beta}A_0$ como $B_\beta \in \{v_\beta : A_0\} \equiv S^{v_\beta}_{B_\beta}A_0$. Las conectivas y cuantificadores pueden ser denotados mediante

[7]Esta es la notación de Andrews para la teoría de tipos de Church, donde la lectura de los tipos se realiza de derecha a izquierda. Mantendremos esta notación en todos los stsitemas que se presentan, exceptuando Alonzo, ya que, como veremos, guía el orden de lectura de los tipos explícitamente con el símbolo $\rightarrow$. Igualmente, emplearemos la notación de Andrews $\circ$ y $\imath$ en vez de 0 y 1 para mantener cierta coherencia en los símbolos en todo el trabajo.

la asignación de tipos. Así, la negación tiene tipo $(\circ\circ)$ porque va de valores de verdad a valores de verdad. La conjunción y la disyunción, en cambio, tienen tipo $(\circ\circ\circ)$ porque son funciones que toman valores de verdad y devuelven funciones de valores de verdad en valores de verdad. El procedimiento anteriormente descrito recibe el nombre de cálculo λ, para muchos, la mejor aportación de Alonzo Church. La base de este cálculo es el concepto de función y que se permita que una función se anide a otras, con lo cual, está perfectamente preparado para expresar recursión. En general, se distingue $F(x)$ y $\lambda F(x)$. Por ejemplo, habitualmente al pensar en la función (x^2+x) mayor que 1000, imaginamos un x concreto que cumpla esa condición. Sin embargo, Church distingue esto de la función recursiva primitiva $\lambda x(x^2+x)$ la cual lleva a cada x al valor x^2+x mayor que 1000. Este cálculo es tan adecuado y compacto que John Mc Carthy lo convirtió en un lenguaje de programación.

2.1.2. Fundamentos de $\mathcal{Q}_0$

En esta sección se presentaran los principios del sistema lógico $\mathcal{Q}_0$ así como algunas definiciones. El sistema lógico $\mathcal{Q}_0$ de teoría de tipos fue propuesto por Peter B. Andrews como una simplificación de la teoría de tipos propuesta por Alonzo Church, el cual desarrolló una segunda versión después de que Russell presentara con Whitehead la primera teoría de tipos en *Principia Mathematica* [17].

En primer lugar, veamos el principio de inducción en la construcción de una fórmula bien formada del lenguaje. Sea R una propiedad de las fórmulas y $R(A)$ correpondiente a decir que A tiene la propiedad R. Supongamos que:
(1) Cada fórmula que consta de una sola variable o constante por sí sola tiene la propiedad R.
(2) Siempre que $A_{\alpha\beta}$ sea una $\text{fbf}_{(\alpha\beta)}$ y B_β sea una fbf_β y $R(A_{\alpha\beta})$ y $R(B_\beta)$, entonces $R([A_{\alpha\beta}B_\beta])$.
(3) Siempre que x_β sea una variable y $R(A_\alpha)$, entonces $R([\lambda x_\beta A_\alpha])$. Entonces, cada fbf de $\mathcal{Q}_0$ tiene la propiedad R.

Antes de entrar en la definición de derivabilidad es importante mencionar que el sistema posee una única regla de inferencia, la regla R. Esta regla nos dice que de C y $A_\alpha = B_\alpha$ se infiere el resultado de reemplazar una aparición de A_α en C por una ocurrencia de B_α, siempre que la aparición de A_α en C no esté inmediatamente precedida por λ [16].

Ahora bien, cuando H sea un conjunto finito de fbf_o, escribiremos $H \vdash A$ como abreviatura de la frase "A es derivable del conjunto de hipótesis H". Una prueba de A de un conjunto de hipótesis H consiste en dos secuencias finitas S_1 y S_2 de fbf_o tal que S_1 es una prueba en $\mathcal{Q}_0$ y A es el último miembro de la secuencia S_2. Cada fbf D^i en S_2 satisface, al menos, una de las siguientes condiciones:

(a) D^i está en H.
(b) D^i está en S_1.
(c) D^i se obtiene de la regla R' de fbf D^j y D^k las cuales preceden a D^i en la secuencia.

La regla R' es una pequeña extensión de la regla R, esta nos dice que si $H \vdash A_\alpha = B_\alpha$, $H \vdash C$ y D se obtiene de C reemplazando una aparición de A_α en C por una de B_α. Entonces $H \vdash D$ ya que la aparición de A_α en C no es una aparición de una variable inmediatamente precedida de λ y tampoco es parte de $(\lambda x_\beta E_\gamma)$ de C, donde x_β es libre en un miembro de H y libre en $(A_\alpha = B_\beta)$. Cabe resaltar que a partir de estas reglas pueden aparecer muchas otras reglas derivadas de inferencia como la eliminación del cuantificador universal, la generalización universal, el modus ponens o la regla de casos a partir de sustitución. Así pues, podemos darnos cuenta de que el sistema de inferencia de $\mathcal{Q}_0$ favorece mucho la implementación informática debido a su sencillez, pues con una única regla podemos inferir muchas otras que consideramos imprescindibles. A modo de ejemplo veamos algunas de las reglas de inferencia anteriores para plasmar su comportamiento.

Regla de sustitución (Sub): Si $H \vdash B$, entonces $H \vdash S^{x^1_{\alpha_1} \ldots x^n_{\alpha_n}}_{A^1_{\alpha_1} \ldots A^n_{\alpha_n}} B$, siempre que las variables $x^1, ..., x^n$ sean distintas una de la otra y distintas de todas las variables libres de las fbf de H. Además $A^i_{\alpha_i}$ sea libre para $x^i_{\alpha_i}$ en B para toda i tal que $1 \leq i \leq n$.
Demostración del caso para $n = 1$:

1) $H \vdash B$	Premisa
2) $H \vdash \forall x_\alpha B$	(Gen)
3) $H \vdash S^{x_\alpha}_{A_\alpha} B$	$\forall I$

Las reglas que se han empleado en esta demostración son:
Regla de Generalización universal: Si $H \vdash A$ entonces $H \vdash \forall x_\alpha A$ siempre que x_α no sea libre en cualquier fbf en H.
Regla de eliminación del cuantificador universal ($\forall I$): Si $H \vdash \forall x_\alpha B$, entonces $H \vdash S^{x_\alpha}_{A_\alpha}$ siempre que A_α sea libre para x_α en B. Estas dos reglas son también reglas derivadas, sin embargo, por una cuestión de extensión no serán demostradas aquí.
Otra regla que merece demostración es el conocido Modus ponens.
Modus ponens (MP): Si $H \vdash A$ y $H \vdash A \supset B$, entonces $H \vdash B$.
Demostración:

1) $H \vdash A \supset B$	Premisa
2) $H \vdash A = T$	Regla T
3) $H \vdash T \supset B$	Regla R'
3) $H \vdash B$	Regla R.

En esta demostración, se han empleado las regla R y las reglas derivadas R' y T. Las reglas R y R' son ya conocidas. La Regla T nos dice que: $H \vdash A$ si y solo si $H \vdash T = A$ o, similarmente, $H \vdash A$ si y solo si $H \vdash A = T$. Nuevamente, omitiremos la demostración de esta regla.

Adicionalmente, para entender correctamente las reglas de inferencia es necesario conocer correctamente el concepto de tautología. Así pues, diremos que una fbf A es una tautología si y solo si A es proposicional y $\mathcal{V}_\varphi A = T$ para todas las asignaciones de valores de verdad de φ. Así pues, si A es una tautología, entonces $\vdash A$; esto es, A es siempre verdadera bajo cualquier interpretación.

2.1.3. Semántica de $\mathcal{Q}_0$

Antes de terminar con la exposición de $\mathcal{Q}_0$, es conveniente definir algunas de las nociones semánticas más fundamentales. En primer lugar, llamamos interpretación de $\mathcal{Q}_0$ a una estructura $\langle \{D_\alpha\}_\alpha, \mathcal{I}\rangle$. $\{D_\alpha\}_\alpha$ es una colección de conjuntos no vacíos D_α, uno para cada símbolo de tipo α tal que $D_o = \{T, F\}$ son los valores designados de verdad. La interpretación $\mathcal{I}$ mapea cada constante de tipo α de $\mathcal{Q}_0$ a un elemento de D_α tal que $\mathcal{I}Q_{0\alpha\alpha}$ es la relación identidad en D_α y $\mathcal{I}_{l_{\imath(o\imath)}}$ es alguna función que mapea cada elemento de $D_{o\imath}$ a su único miembro de $D_\imath$.

Una asignación a un conjunto $\{D_\alpha\}_\alpha$ es una función φ con dominio en el conjunto de variables de $\mathcal{Q}_0$ tal que para cada variable x_α, $\varphi x_\alpha \in D_\alpha$.
De este modo, un modelo es una interpretación $\mathcal{M} = \langle \{D_\alpha\}_\alpha, \mathcal{I}\rangle$ para $\mathcal{Q}_0$ si y solo si hay una función binaria $\mathcal{V}^\mathcal{M}$ tal que para cada asignación φ y fórmula bien formada A_α, $\mathcal{V}^\mathcal{M}_\varphi A_\alpha \in D_\alpha$.

Además, se satisfacen las siguientes condiciones para cada asignación φ y toda fbf [16]:
(a) $\mathcal{V}^\mathcal{M}_\varphi A_\alpha = \varphi x_\alpha$.
(b) $\mathcal{V}^\mathcal{M}_\varphi A_\alpha = \mathcal{I}A_\alpha$ si A_α es una constante primitiva.
(c) $\mathcal{V}^\mathcal{M}_\varphi(A_{\alpha\beta}B_\beta) = (\mathcal{V}^\mathcal{M}_\varphi A_{\alpha\beta})(\mathcal{V}^\mathcal{M}_\varphi B_\beta)$(el valor de la función $\mathcal{V}^\mathcal{M}_\varphi A_{\alpha\beta}$)
(d) $\mathcal{V}^\mathcal{M}_\varphi(\lambda x_\alpha B_\beta)$ = una función de D_α a D_β cuyos valores para cada argumento $z \in D_\alpha$ es $\mathcal{V}^\mathcal{M}_{\varphi:x_\alpha/z} B_\beta$.

Si una interpretación es un modelo $\mathcal{M}$, la función $\mathcal{V}^\mathcal{M}$ es unívoca.
De este modo, diremos que φ satisface A en $\mathcal{M}$ si y solo si $\mathcal{V}^\mathcal{M}A = T$. También diremos que A es satisfacible en $\mathcal{M}$ si y solo si hay al menos una asignación que satisface A en $\mathcal{M}$. Asimismo, A es válida en $\mathcal{M}$ si y solo si para cada asignación φ en $\mathcal{M}$, $\mathcal{V}^\mathcal{M}A = T$.

Por otra parte, las nociones de consistencia, corrección y completud son fundamentales en todo sistema lógico. De hecho, el sistema $\mathcal{Q}_0$ es consistente y, para cada conjunto de fbfs$_o$ que

tiene un modelo, este también es consistente. Respecto a la corrección, cada teorema de $\mathcal{Q}_0$ es válido en un sentido general. Además, si G es un conjunto de fbfs_o y $\mathcal{M}$ es un modelo para G y $G \vdash A$, entonces $\mathcal{M} \vDash A$; esto es, el teorema de corrección. La demostración de este teorema pasa por la preservación de validez en la regla R en cada modelo de $\mathcal{Q}_0$, además de comprobar que los axiomas de $\mathcal{Q}_0$ son válidos en un sentido general. El teorema de completud se expone de forma análoga a las que hemos visto durante el curso. Diremos que un conjunto H de sentencias de $\mathcal{Q}_0$ es completo si y solo si para cada sentencia A se tiene que $H \vdash A$ o bien $H \vdash \neg A$. Por último, diremos que H es extensionalmente completo si y solo si para cada sentencia de la forma $(A_{\alpha\beta} = B_{\alpha\beta})$ hay una C_β tal que $H \vdash (A_{\alpha\beta}C_\beta = B_{\alpha\beta}C_\beta) \supset (A_{\alpha\beta} = B_{\alpha\beta})$.

2.2. Alonzo

Al introducirnos a la teoría de tipos de Church hemos visto que el sistema $\mathcal{Q}_0$ es simple, elegante y con un alto poder expresivo. Además, brinda la oportunidad de ser aplicable en computación y de ser una alternativa a la formalización de las matemáticas debido a su naturalidad. En esta sección veremos una alternativa de la teoría de tipos que se deriva de la de Church: Alonzo. Esta sistema tiene la particularidad de admitir expresiones indefinidas, lo que supone una ventaja para la computación y el razonamiento matemático. Así como en las matemáticas pueden escribirse expresiones como $\frac{1}{0}$ o bien $\sqrt{-2}$, Alonzo permite este tipo de expresiones y les asocia el significado "indefinido". Son expresiones perfectamente legítimas pero que no denotan nada. Por lo tanto, es de gran utilidad para aquellos que se dedican a estudiar los fundamentos de las matemáticas o para aquellos que se dedican a un sector más aplicado como la ingeniería de datos.

En Alonzo, las expresiones atómicas son siempre definidas, sin embargo, las expresiones compuestas pueden ser indefinidas de tal manera que una aplicación de función $f(a)$ es indefinida si f es indefinida, a es indefinida o $a \notin dom(f)$. Las fórmulas, por otra parte, toman los valores designados de verdad clásicos. Cabe resaltar que cuando una expresión es indefinida, afecta siempre a la totalidad de la fórmula. Esto es, una fórmula que contenga al menos una expresión indefinida será siempre falsa. Por este motivo, veremos que se incluye la casi igualdad $\sim$. De esta forma, la expresión $\frac{1}{0} = \frac{1}{0}$ es falsa, pero la expresión $\frac{1}{0} \sim \frac{1}{0}$ es verdadera [18].

Similarmente a $\mathcal{Q}_0$, en este sistema los tipos clasifican las expresiones por su valor y controlan la formación de expresiones. De este modo, puede entenderse como una teoría de funciones con cuantificación sobre estas. Así pues, se extiende la FOL pero se basa también en algunos de sus principios. Existen dos versiones del sistema Alonzo: *Core* Alonzo y *Full* Alonzo. La primera es una versión minimalista basada en igualdad, aplicación de funciones y abstracción de funciones. La segunda se extiende con maquinaria para descripciones definidas, conjuntos, tuplas y listas.

2.2.1. *Core* Alonzo

Core Alonzo, de hecho, Alonzo en general, consiste en tres conjuntos infinitamente contables de símbolos: $\mathcal{S}_{bt}, \mathcal{S}_v, \mathcal{S}_c$. Estos son los símbolos de tipos, los símbolos de variables y los símbolos de constantes. Cabe resaltar que a estos símbolos se añaden también los del conjunto $\{\circ, \rightarrow, (,), \lambda, .\}$. Asumimos que estos cuatro conjuntos son disjuntos.

A modo de esquema, los símbolos que encontraremos son de la forma siguiente [19]:
$\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}...$ son variables sintácticas que ranguean sobre miembros de $\mathcal{S}_{bt}, \mathcal{S}_v, \mathcal{S}_c$.
$\alpha, \beta, \gamma, \delta...$ son variables sintácticas que ranguean sobre los tipos.
$\mathbf{f}_\alpha, \mathbf{g}_\alpha...$ son variables sintácticas que ranguean sobre variables de tipo α.
$\mathbf{c}_\alpha, \mathbf{c}_\alpha...$ son variables sintácticas que ranguean sobre constantes de tipo α.
$\mathbf{A}_\alpha, \mathbf{B}_\alpha...$ son variables sintácticas que ranguean sobre expresiones de tipo α; esto es, predicados, conjuntos o propiedades.

Además, como en todo sistema lógico, existen una serie de reglas de formación.
Para la construcción del conjunto de tipos $\mathcal{T}$ [19]:
T1 (Valores de verdad): $\circ$ es un tipo.
T2 (Tipos base): $\mathbf{a}$ es un tipo.
T3 (Tipos de funciones)[8]: $(\alpha \rightarrow \beta)$ es un tipo.

Para el conjunto $\mathcal{E}$, el cual denota las expresiones de Alonzo [19]:
E1 (Variables): $(\mathbf{x} : \alpha)$ es una expresión de tipo α.
E2 (Constantes): $\mathbf{c}_\alpha$ es una expresión de tipo α.
E3 (Igualdad): $(\mathbf{A}_\alpha = \mathbf{B}_\alpha)$ es una expresión de tipo $\circ$.
E4 (Aplicación de funciones)[9]: $(\mathbf{F}_{(\alpha\rightarrow\beta)}\mathbf{A}_\alpha)$ es una función de tipo β.
E5 (Abstracción de funciones): $(\lambda \mathbf{x} : \alpha.\mathbf{B}_\beta)$ es una expresión de tipo $(\alpha \rightarrow \beta)$; esto es, mapea variables de tipo α a expresiones de tipo β. Esto es, la descripción de una λ-expresión.
Asimismo, una fórmula es una expresión de tipo $\circ$.

Para finalizar con la sintaxis, denotamos el lenguage de Alonzo (*Core* y *Full*) por el par $L = \langle \mathcal{B}, \mathcal{C} \rangle$, donde $\mathcal{B}$ es un conjunto finito de tipos base y $\mathcal{C}$ es un conjunto de constantes. Claramente, de este modo obtenemos que $\mathcal{T}(L) \subseteq \mathcal{T}$ y que $\mathcal{E}(L) \subseteq \mathcal{E}$. Además, $\mathcal{B}$ y $\mathcal{C}$ pueden ser conjuntos vacíos, pero no pueden serlo $\mathcal{T}(L)$ ni $\mathcal{E}(L)$ ya que siempre $\circ \in \mathcal{T}(L)$.

[8] Sin pérdida de generalidad, se asumen algunas convenciones para suavizar la notación, de este modo, $(\alpha \rightarrow (\beta \rightarrow \gamma))$ puede escribirse simplemente como $\alpha \rightarrow \beta \rightarrow \gamma$.

[9] En este caso, relajamos la notación asociando hacia la izquierda, por ejemplo, $((\mathbf{G}_{\alpha\rightarrow\beta\rightarrow\gamma}\mathbf{A}_\alpha)\mathbf{B}_\beta)$ puede escribirse como $\mathbf{G}_{\alpha\rightarrow\beta\rightarrow\gamma}\mathbf{A}_\alpha\mathbf{B}_\beta$

Atendiendo a la semántica de Alonzo, tenemos en primer lugar que el marco de un lenguaje L es una colección $\mathcal{D} = \{D_\alpha | \alpha \in \mathcal{T}(L)\}$ de dominios (conjuntos) no vacíos de valores tales que $D_\circ = \{F, T\}$ y $D_{\alpha\to\beta}$ es un conjunto de algunas funciones de D_α a D_β para $\alpha, \beta \in \mathcal{T}(L)$. Esto es, la lógica es bivaluada e incluye funciones. Diremos que un marco es completa cuando el conjunto $D_{\alpha\to\beta}$ contiene todas las funciones. Así pues, denotaremos una interpretación de L con el par $\mathcal{I} = \langle \mathcal{D}, I \rangle$, donde I mapea cada constante en $\mathcal{C}$ de tipo α a un elemento de D_α. Nótese que este par, además de denotar una interpretación, es una estructura, tal y como cabe esperar. En cuanto a las asignaciones, podemos decir que son funciones ϕ cuyo dominio es el conjunto de variables de L tal que $\phi(\mathbf{x} : \alpha) \in D_\alpha$ para cada variable $(\mathbf{x} : \alpha)$ de L. Asimismo, dada una asiganación ϕ, una variable $(\mathbf{x} : \alpha)$ y $d \in D_\alpha$, sea $\phi[(\mathbf{x} : \alpha) \mapsto d]$ la asignación ψ en $\mathcal{D}$ tal que $\psi(\mathbf{x} : \alpha) = d$ y $\psi(\mathbf{y} : \beta) = \phi(\mathbf{y} : \beta)$ para todas las variables $(\mathbf{y} : \beta)$ distintas de $(\mathbf{x} : \alpha)$.

Por lo que respecta a los modelos, diremos que M es un modelo general para L si hay una función de valuación parcial binaria $\mathcal{V}^M$ tal que para todas las asignaciones y expresiones de L o bien $\mathcal{V}^M_\phi(\mathbf{C}_\gamma) \in D_\gamma$ o $\mathcal{V}^M_\phi(\mathbf{C}_\gamma)$ es indefinido, o bien se satisfacen una serie de condiciones:
$\mathcal{V}^M_\phi(\mathbf{x} : \alpha) = \phi(\mathbf{x} : \alpha)$.
$\mathcal{V}^M_\phi(\mathbf{c}_\alpha = I(\mathbf{c}_\alpha)$.
$\mathcal{V}^M_\phi(\mathbf{A}_\alpha = \mathbf{B}_\alpha) = T$ si $\mathcal{V}^M_\phi(\mathbf{A}_\alpha)$ y $\mathcal{V}^M_\phi(\mathbf{B}_\alpha)$ definidos y $\mathcal{V}^M_\phi(\mathbf{A}_\alpha) = \mathcal{V}^M_\phi(\mathbf{B}_\alpha)$; falso en otro caso.
$\mathcal{V}^M_\phi(\mathbf{F}_{\alpha\to\beta}\mathbf{A}_\alpha) = \mathcal{V}^M_\phi(\mathbf{F}_{\alpha\to\beta})(\mathcal{V}^M_\phi(\mathbf{A}_\alpha))$ si $\mathcal{V}^M_\phi(\mathbf{F}_{\alpha\to\beta})$ es definido y $\mathcal{V}^M_\phi(\mathbf{A}_\alpha)$ es definido. En otro caso $\mathcal{V}^M_\phi(\mathbf{F}_{\alpha\to\beta}\mathbf{A}_\alpha) = F$ si $\beta = \circ$ y $\mathcal{V}^M_\phi(\mathbf{F}_{\alpha\to\beta}\mathbf{A}_\alpha)$ será indefinido si $\beta \neq \circ$. Esto es, el significado "indefinido" es infeccioso cuando $\beta \neq \circ$.
$\mathcal{V}^M_\phi(\lambda\mathbf{x} : \alpha.\mathbf{B}_\beta)$ es la función $f \in D_{\alpha\to\beta}$ tal que para cada $d \in D_\alpha$ tenemos que $f(d) = \mathcal{V}^M_{\phi[(\mathbf{x}:\alpha)\mapsto d]}(\mathbf{B}_\beta)$ si $\mathcal{V}^M_{\phi[(\mathbf{x}:\alpha)\mapsto d]}(\mathbf{B}_\beta)$ es definido. En caso contrario, $f(d)$ será también indefinido.
Por otra parte, diremos que M es un modelo estándar si $\mathcal{D}$ es completo. Cabe resaltar entonces que todo modelo estándar es un modelo general pero no a la inversa. En definitiva, el lenguaje de *Core* Alonzo dispone de dos semánticas distintas.

Para acabar con la semántica, podemos comprobar que las nociones de satisfacibilidad, validez y consecuencia semántica se comportan de la forma habitual.
ϕ satisface $\mathbf{A}_\circ$ en M ($M \vdash_\phi \mathbf{A}_\circ$) si $\mathcal{V}^M_\phi(\mathbf{A}_\circ) = T$. Asimismo, $\mathbf{A}_\circ$ es satisfacible en M si $M \vdash_\phi \mathbf{A}_\circ$) para alguna asignación ϕ. Por el contrario, diremos que $\mathbf{A}_\circ$ es válida en M si $M \vdash_\phi \mathbf{A}_\circ$) para alguna asignación ϕ; esto es, M es un modelo de $\mathbf{A}_\circ$.
Finalmente, diremos que $\mathbf{A}_\circ$ es consecuencia semántica de un conjunto de fórmulas Γ ($\Gamma \vdash \mathbf{A}_\circ$ si $M \vdash \Gamma$ implica $M \vdash \mathbf{A}_\circ$ para todo modelo M y toda asignación ϕ.

En último lugar, podemos ver cómo se definen algunos operadores en Alonzo. El operador $T_\circ$ denota verdad y corresponde a la fórmula $(\lambda x : \circ.x) = (\lambda x : \circ.x)$. Similarmente, el de falsedad $F_\circ$ corresponde a la fórmula $(\lambda x : \circ.T_\circ) = (\lambda x : \circ.x)$. Además, pueden definirse también los operadores booleanos correspondientes a las conectivas habituales. Por ejemplo. $\wedge_{\circ\to\circ\to\circ}$ denota la conjunción por la fórmula $(\lambda x : \circ.\lambda y : \circ.(\lambda g : \circ \to \circ \to \circ.gT_\circ T_\circ) = (\lambda g : \circ \to \circ \to \circ.gxy))$. Así pues, la expresión $(\mathbf{A}_\circ \wedge \mathbf{B}_\circ)$ y la expresión $\wedge_{\circ\to\circ\to\circ}\mathbf{A}_\circ\mathbf{B}_\circ$ son equivalentes. Este proceso también puede realizarse para los cuantificadores existencial y universal. Adicionalmente, tenemos algunos operadores extra. $\mathbf{A}_\alpha \uparrow$ significa que $\mathbf{A}_\alpha$ es indefinido. Contrariamente, con $\mathbf{A}_\alpha \downarrow$ expresamos que $\mathbf{A}_\alpha$ es definido, equivalentemente $\neg(\mathbf{A}_\alpha \uparrow)$. Esto es, $\mathbf{A}_\alpha \downarrow$ es una abreviatura de la expresión $((\lambda \mathbf{x} : \alpha.T_\circ)\mathbf{A}_\alpha)$, la cual puede leerse como "$\mathbf{A}_\alpha$ converge" o "$\mathbf{A}_\alpha$ denota". Además, tenemos la casi igualdad, expresada como $\mathbf{A}_\alpha \sim \mathbf{B}_\alpha$ que corresponde a la fórmula $(\mathbf{A}_\alpha \downarrow \vee \mathbf{B}_\alpha \downarrow) \supset \mathbf{A}_\alpha = \mathbf{B}_\alpha$.

2.2.2. *Full* Alonzo

La forma en la que extendemos *Core* Alonzo para dar con *Full* Alonzo es añadiendo nuevos constructores a las definiciones de tipo y expresión con nuevas cláusulas.

En primer lugar, los subtipos de los tipos α son cualquier expresión de tipo $\{\alpha\}$, esto equivale al tipo $(\alpha \to \circ)$. Así pues, un subtipo $\mathbf{A}_{\{\alpha\}}$ denota un subconjunto de D_α. La cláusula que añadimos a las expresiones corresponde a la que permite la aparición de descripciones definidas y es como sigue: $(\mathrm{I}\mathbf{x} : \alpha.\mathbf{B}_\circ)$ es una expresión de tipo α donde $\alpha \neq \circ$. Además, añadimos también una cláusula para los modelos que nos dice que $\mathcal{V}^M_\phi(\mathrm{I}\mathbf{x} : \alpha.\mathbf{B}_\circ)$ es el $d \in D_\alpha$ tal que $\mathcal{V}^M_{\phi[(\mathbf{x}:\alpha)\mapsto d]}(\mathbf{B}_\beta) = T$ si hay exactamente un d. En otro caso, será indefinido.

Para las tuplas añadimos la siguiente maquinaria [20]:
T4 (Producto de tipos): $(\alpha \times \beta)$ es un tipo.
F3: $D_{\alpha\times\beta} = D_\alpha \times D_\beta$.
E7 (Par ordenado): $(\mathbf{A}_\alpha, \mathbf{B}_\beta)$ es una expresión de tipo $\alpha \times \beta$.
V7: $\mathcal{V}^M_\phi((\mathbf{A}_\alpha, \mathbf{B}_\beta)) = (\mathcal{V}^M_\phi(\mathbf{A}_\alpha), \mathcal{V}^M_\phi(\mathbf{B}_\beta))$.

Finalmente, para las listas, añadimos las condiciones [20]:
T5 (Tipo de una lista): $[\alpha]$ es un tipo.
F4: $D_{[\alpha]} = D^*_\alpha$.
E8 (Construcción de una lista): $(\mathbf{A}_\alpha :: \mathbf{B}_{[\alpha]})$ es una expresión de tipo $[\alpha]$.
V8: $\mathcal{V}^M_\phi(\mathbf{A}_\alpha :: \mathbf{B}_{[\alpha]}) = \mathcal{V}^M_\phi(\mathbf{A}_\alpha) :: \mathcal{V}^M_\phi(\mathbf{B}_{[\alpha]})$.

2.3. El sistema lógico PF

En la teoría de tipos de Church se asume que las funciones son totales, es decir, aplican sobre todo el dominio. Sin embargo, en el sistema PF veremos que esto no es necesariamente así ya que podemos tener también funciones parciales, las cuales aplican a un subconjunto del dominio. Esto sucede a menudo de forma natural en las matemáticas, por consiguiente, los cambios en PF son también muy naturales para aquellos que quieran aplicar la teoría ed tipos a las matemáticas o la computación. El sistema lógico PF parte de la teoría de tipos de Church con pequeñas modificaciones en la semántica de modelos generales de Henkin que habilitan la existencia de funciones parciales. Cabe resaltar que estas modificaciones no alteran la completitud de la lógica.

2.3.1. Funciones parciales y términos no denotativos

En lógica, los términos se usan para describir valores en las estructuras matemáticas y las fórmulas se usan para realizar aserciones sobre estos valores. En la semántica clásica se asume la existencia y la denotación de los términos en cualquier caso. Sin embargo, nos adentramos en una problemática al intentar incluir las funciones parciales en el marco de la lógica ya que conducen a la violacioón del supuesto de existencia. Veamos esto con algo más de profundidad. Consideremos un término construido a partir de expresiones más simples mediante la aplicación de una expresión que denota una función a una expresión que denota el argumento de una función. Si el argumento se encuentra fuera del dominio de la función entonces el término no estaría bien definido y no tendría denotación natural. Resumiendo, el uso de funciones parciales conduce a términos que no denotan. Un ejemplo, “La playa de Madrid” es una descripción definida no denotativa. Existen diferentes aproximaciones para lidiar con las problemáticas de la no denotación de términos y la cuantificación, cada de estas aproximaciones da lugar a una semántica distinta y, por ende, a una lógica distinta. No obstante, veremos solamente algunas de las vías para lidiar la problemática ya que son las que plantean un razonamiento más natural de las matemáticas y la computación. Finalmente, veremos cómo estas vías conducen a una última en la que se desarrolla el sistema PF, cuya notación es eficiente, expresiva y fiel a la práctica sin trastornar en exceso el marco de la lógica.

Una de las aproximaciones es entender las expresiones no denotativas como términos mal formados. Esta aproximación tiene algunos defectos ya que un término puede ser bien o mal formado dependiendo del contexto de interpretación. Además, no existe un algoritmo operacional que pueda determinar cuando un término será bien o mal formado. Presumiblemente, los términos mal formados no son expresiones legítimas de manipular y, aún así, en matemáticas se emplean frecuentemente hasta que no se determina su valor (Farmer, 1990, p.1271-1275) [21].

Otra de las vías que podemos seguir es que todas las funciones se entiendan como totales pero que algunas de ellas tengan valores inespecificados, es decir, la imagen es conocida para los elementos del dominio y desconocida para los que quedan fuera [21]. En la división de racionales, por ejemplo, la función $f(23, 0)$ sería inespecificada, por lo tanto, no podemos decir nada sobre su valor excepto que será un racional. Esta vía es ventajosa en cuanto al marco de la lógica ya que las funciones siguen siendo totales. No obstante, no cumple nuestros requisitos ya que viendo la solución con ojos de matemático resulta bastante antinatural que $\frac{23}{0} = \frac{45}{98}$, por ejemplo.

Una aproximación que se podría tomar es la lógica multivariada [21], donde la división racional sería total si se define bajo $\mathbb{Q} \times \mathbb{Q}^{-}$, donde $\mathbb{Q}^{-} = \mathbb{Q} - \{0\}$ con los tipos adecuados. Sin embargo, esta forma natural de convertir las funciones parciales en totales en el contexto de la lógica multivariada solo es viable en algunas ocasiones. Por ejemplo, la función $f(x, y) = \sqrt{x - y}$ no tiene un dominio de la forma $D_x \times D_y \subseteq \mathbb{R}$. Así pues, para cada función g_a definida por $g_a(x) = f(x, a)$ se tiene un dominio distinto para cada a, así que cada vez que se construye una función como g_a se requiere un nuevo símbolo de tipo, lo que provoca la proliferación de los símbolos de tipos.

Con la visión de un informático, lo natural al pensar en funciones parciales y términos no denotativos es asociarles el valor *error*. La ventaja de esta aproximación es que mantiene la coherencia ya que si $a = error$, entonces $f(a) = error$, aunque no siempre es estricto ya que en matemáticas y computación podemos agregar condiciones al dominio de aplicación. Por lo tanto, una función con tres argumentos podría tomar un valor definido incluso si uno de ellos tiene valor *error*. Por otra parte, si tratamos de buscar una aproximación satisfactoria para matemáticas y computación, no es la mejor opción ya que cuantificar sobre un valor *error* en matemáticas como si fuera un valor clásico no tiene demasiado sentido. Podríamos cuantificar sobre un valor $indefinido$ con un trato distinto a los valores clásicos, pero la realidad matemática cuantifica sobre todo valor, así que no es una alternativa del todo fiel.

En último lugar, podemos optar por una valuación parcial para los términos y una valuación total para las fórmulas [21]. En esta aproximación un término denota un valor solamente si todos sus subtérminos denotan un valor, mientras que una fórmula denota siempre en valores de verdad de forma que una fórmula atómica será falsa si cualquier término con apariciones en ella no denota. La primera regla propaga la no denotación en términos y la segunda norma garantiza que todas las fórmulas tengan valores estándar. Estas reglas parecen representar de forma bastante fiel la forma en la que las matemáticas piensan acerca de los términos no denotativos y, a su vez, no complica demasiado la implementación computacional.

2.3.2. Lenguaje formal de PF

El lenguaje formal de PF es esencialmente el mismo que el de $\mathcal{Q}_0$ y Alonzo, por lo tanto, no dedicaremos la misma extensión a esta sección.

El conjunto de los tipos se define inductivamente siguiendo las reglas de construcción descritas en *Core* Alonzo. Sin embargo, podemos relajar la notación para los tipos de funciones escribiendo simplemente $(\alpha\beta)$ en vez de $(\beta \to \alpha)$, a diferencia de Alonzo y tal como hemos descrito en $\mathcal{Q}_0$. Además, en PF disponemos de funciones totales pero también parciales. Así pues, $(\alpha\beta) \in \mathcal{T}^\imath$ denota el tipo de las funciones parciales y totales de elementos de tipo β a elementos de tipo α. En cambio, $(\alpha\beta) \in \mathcal{T}^\circ$ solamente denota el tipo de funciones totales. Siguiendo este razonamiento, el tipo de los conjuntos con elementos de tipo α es $(\circ\alpha)$, como es habitual.

Los símbolos primitivos de PF son los mismos que para Alonzo, incluyendo la constante lógica $=_{((\circ\alpha)\alpha)}$ para cada tipo α.

Una expresión de PF de tipo $\alpha \in \mathcal{T}$ se define inductivamente como una secuencia finita de símbolos primitivos de la misma forma que en Alonzo. No obstante, agilizaremos la notación según el criterio de $\mathcal{Q}_\prime$ para la omisión de "$\to$". Además, omitiremos también los símbolos ":" y "." usando subíndices. Por ejemplo, la norma **E5** para la abstracción de funciones quedaría tal que así: si $\mathbf{x}_\alpha \in \mathcal{S}_v$ y $\mathbf{M}_\beta \in \mathcal{E}$, entonces $(\lambda\mathbf{x}_\alpha\mathbf{M}_\beta)$ es una expresión de tipo $(\beta\alpha)$.

Finalmente, diremos que una expresión es de clase (no de tipo) $\imath$ o de clase $\circ$ si es una expresión de tipo $\alpha \in \mathcal{T}^\imath$ o de tipo $\alpha \in \mathcal{T}^\imath$, respectivamente. Intuitivamente, las expresiones de clase $\imath$ sirven como funciones y términos, mientras que los de clase $\circ$ sirven como predicados y fórmulas. A pesar de ser expresiones con usos distintos, se tratan sintácticamente igual.

2.3.3. Semántica de PF

Dado que las definiciones de marco e interpretación son análogas a las de Alonzo y $\mathcal{Q}_0$, es deseable dedicar este apartado a exponer los principios que motivan la semántica de PF. Así pues, no dedicaremos tampoco este espacio a definir nuevamente los conceptos de validez, satisfacibilidad, consecuencia semántica, etc.

PRINCIPIO 1 (parcialidad): Las funciones parciales pueden denotarse por expresiones de clase $\imath$.
PRINCIPIO 2 (valores de verdad): Las expresiones de tipo $\circ$ denotan valores de verdad.
PRINCIPIO 3 (expresiones distintas de la aplicación): Las variables, las constantes y las λ-expresiones siempre tienen una denotación.

PRINCIPIO 4 (aplicaciones que no denotan): Una aplicación $\mathbf{F}_{\alpha\beta}\mathbf{A}_{\beta}$ de clase $\imath$ tiene una denotación si y solo si $\mathbf{F}_{\alpha\beta}$ y $\mathbf{A}_{\beta}$ tienen denotaciones f y a, respectivamente, además, f es definida en a.
PRINCIPIO 5 (predicado de expresiones no denotativas): Una aplicación $\mathbf{F}_{\circ\alpha}\mathbf{A}_{\alpha}$ es falsa si la expresión $\mathbf{A}_{\alpha}$ no denota.

Evidentemente, cada uno de los principios sobre los que descansa la semántica tiene una razón de ser [21]. Los principios 1 y 2 establecen una diferencia semántica entre las clases $\imath$ y $\circ$ ya que la lógica permite la funciones parciales sin sacrificar la lógica bivaluada. El principio 3 nos dice que solamente las aplicaciones pueden no denotar, tal como podríamos intuir por la aproximación utilizada en la construcción de PF. El principio 4 nos dice que la no denotación se propaga sobre términos de clase $\imath$ y que la aplicación de una expresión que denota una función parcial a una expresión que denota un valor fuera del dominio de la función es en sí misma no denotativa. El principio 5 dice que la aplicación de un predicado a una expresión no denotativa es siempre falsa.

2.3.4. Cálculo de PF

La lógica de PF es una adaptación de $\mathcal{Q}_0$ que incluye algunas características de Alonzo. Esta lógica se compone de 10 axiomas y una única regla de inferencia [21]. Es importante señalar que 9 de los 10 axiomas son realmente esquemas de axiomas (uno para cada tipo) y que la regla de inferencia corresponde a la regla R de $\mathcal{Q}_0$ con una ligera modificación.

AXIOMA 1 (Valores de verdad): $g_{\circ\circ}T_{\circ} \wedge g_{\circ\circ}F_{\circ} = \forall x_{\circ}(g_{\circ\circ}x_{\circ})$.
AXIOMA 2 (Ley de Leibniz): $(x_{\alpha} = y_{\alpha}) \supset (h_{\circ\alpha}x_{\alpha} = h_{\circ\alpha}y_{\alpha})$.
AXIOMA 3 (Extensionalidad): $(f_{\alpha\beta} = g_{\alpha\beta}) = \forall x_{\beta}(f_{\alpha\beta}x_{\beta} \sim g_{\alpha\beta}x_{\beta})$.
AXIOMA 4 (Reducción β): $A_{\alpha}\downarrow\supset [(\lambda x_{\alpha}B_{\beta})A_{\alpha} \sim B_{\beta}[^{x_{\alpha}}_{A_{\alpha}}]]$, donde A_{α} es libre para x_{α} en B_{β}.
AXIOMA 5 (Expresiones de clase $\circ$ convergen): $A_{\alpha}\downarrow$.
AXIOMA 6 (Variables convergen): $x_{\alpha}\downarrow$.
AXIOMA 7 (Constantes convergen): $c_{\alpha}\downarrow$.
AXIOMA 8 (λ-expresiones convergen): $(\lambda x_{\alpha}B_{\beta})\downarrow$.
AXIOMA 9 (Propagación de divergencia): $(A_{\alpha\beta}B_{\beta})\downarrow\supset (A_{\alpha\beta}\downarrow \wedge B_{\beta}\downarrow)$.
AXIOMA 10 (Predicado de expresiones divergentes): $B_{\beta}\uparrow\supset [A_{\alpha\beta}B_{\beta} = F_{\alpha}]$.
Regla R: de la expresión $A_{\alpha} \sim B_{\alpha}$ y $C_{\circ}$ inferimos el resultado de reemplazar una aparición de A_{α} en $C_{\circ}$ por una aparición de B_{α} siempre que la aparición de A_{α} en $C_{\circ}$ no sea una aparición de una variable inmediatamente precedida de λ.

Con todo esto, una prueba de una fórmula $A_{\circ}$ en PF es una secuencia finita de fórmulas que acaban en $A_{\circ}$ tal que cada miembro de la secuencia es un axioma o un derivado de él mediante

la regla R. Así pues, un teorema es una fórmula con una prueba en PF. Cabe resaltar también que cada uno de los axiomas da cuenta de alguno de los principios semánticos anteriores. Los dos primeros axiomas se toman de $\mathcal{Q}_0$. Los axiomas 3 y 4 dan cuenta del principio 1 y el principio 5 lo hace del principio 2. Los axiomas 6, 7 y 8 representan al principio 3, mientras que el principio 9 representa al principio 4. Finalmente, el último axioma da cuenta del principio 5.

2.3.5. Metateoria de PF

La metateoria de PF es prácticamente análoga a la de $\mathcal{Q}_0$. Una forma de ver esta analogía es considerando el sistema formal que se obtiene al añadir a los axiomas de PF todas las instancias del esquema de $A_\alpha \downarrow$. Llamemos por ejemplo a este sistema PF'. Es sencillo ver cómo toda fórmula $A_\circ$ es un teorema de PF' si y solo si $A_\circ$ es un teorema de $\mathcal{Q}_0$. Intuitivamente, esto es lo mismo que decir que $\mathcal{Q}_0$ es igual a PF si todas las funciones son totales.

Veamos pues, las diferencias entre PF y PF'. Para ello, podemos probar que PF y PF' no tienen los mismos teoremas. La fórmula $A_\circ = x_\iota y_\iota \downarrow$ es, evidentemente, un teorema de PF'. Sea $\mathcal{M} = \langle \mathcal{D}_\alpha, I \rangle$ un modelo estándar para PF. Dado que $\mathcal{D}_{\iota\iota}$ contiene cada función parcial de $\mathcal{D}_\iota$ a $\mathcal{D}_\iota$, contiene también la función vacío, por lo tanto, $A_\circ$ no es válida en $\mathcal{M}$. Así pues, por el teorema de corrección sabemos que $A_\circ$ no es un teorema de PF. Concretamente, el teorema de corrección nos dice que todo teorema de PF es válido en sentido general y, por lo tanto, también lo es en sentido estándar. Para ver esto, simplemente podemos razonar que cada axioma de PF es válido en sentido general y que la regla R (única) preserva validez en sentido general.

Atendiendo a la completitud, ya sabemos que Henkin desmostró que $\mathcal{Q}_0$ es completo en el sentido que una fórmula es un teorema de $\mathcal{Q}_0$ si y solo si es válida en todo modelo general de $\mathcal{Q}_0$. Dado que $\mathcal{Q}_0$ incluye el poder expresivo de la SOL, se sigue del teorema de incompletitud de Gödel que hay fórmulas válidas para todo modelo estándar en $\mathcal{Q}_0$ pero no son demostrables. Se puede demostrar el teorema de completitud al estilo Henkin para PF. Precisamente, se puede demostrar que cada fórmula $A_\circ$ de PF es un teorema si y solo si $A_\circ$ es válida en sentido general. (Andrews, 2002, p.248-256) [16]. (Farmer, 1990, p.1282-1286) [21].

2.3.6. Operadores de descripción definida

En el contexto de los tipos simples, una función de descripción para tipo α es una función f de tipo $\alpha(\circ\alpha)$ tal que si h es una función de tipo $(\circ\alpha)$ siendo la función característica de un conjunto de un elemento (*singleton*) ($\{a\}$), entonces $f(h) = a$. Las funciones de descripción se clasifican según su comportamiento en funciones características para conjuntos distintos del *singleton*. Por ejemplo, una función de elección es una función de descripción tal que si h es la función de

característica de un conjunto $A \neq \varnothing$, entonces $f(h) \in A$. En concreto, nos interesan las funciones de descripción definida que ya que son las aptas para funciones parciales. Así pues, una función de descripción de tipo α es definida si $f(h)$ es indefinida siempre que h no es la función característica para un *singleton*. De esta forma, una función de descripción definida siempre es no total. Además, cualquier función de descripción definida del tipo que sea para clase $\imath$ puede formalizarse en PF como un operador de descripción definida. Dado que las funciones de descripción definida son funciones parciales, pueden formalizarse en PF de manera mucho más natural que en $\mathcal{Q}_0$.

Sea Γ un conjunto de fórmulas. Un operador de descripción en Γ para tipo α es una expresión cerrada $A_{\alpha(\circ\alpha)}$ tal que $\Gamma \vdash [h_{\circ\alpha}x_\alpha \wedge \forall y_\alpha(h_{\circ\alpha}y_\alpha \supset x_\alpha = y_\alpha)] \supset [x_\alpha = A_{\alpha(\circ\alpha)h_{\circ\alpha}}]$. Esto es, un operador de descripción en Γ denota una función de descripción en Γ. Existen dos aproximaciones para la formalización de funciones de descripción definida como operadores de descripción. La vía indirecta emplea la teoría de funciones totales con valores inespecificados. La vía directa, en cambio, formaliza una función de descripción definida para tipo α diretamente como un operador de descripción parcial $A_{\alpha(\circ\alpha)}$ en Γ. Esta vía es la que emplearemos ya que se ajusta más a nuestros objetivos según los principios semánticos establecidos en la construcción de PF. Además, esta vía puede emplearse en PF pero no en $\mathcal{Q}_0$. Así pues, diremos que un operador de descripción $A_{\alpha(\circ\alpha)}$ en Γ es definido si $\Gamma \vdash \neg\exists x_\alpha(h_{\circ\alpha} = (=_{\circ\alpha\alpha})) \supset A_{\alpha(\circ\alpha)}(h_{\circ\alpha})\uparrow$. Asumamos también que $\mathcal{S}_c$ contiene una constante $d_{\alpha(\circ\alpha)}$ para todo $\alpha \in \mathcal{T}^\imath$ y definamos también $\Delta_\alpha = \{d_{\alpha(\circ\alpha)}(=_{\circ\alpha\alpha} x_\alpha) = x_\alpha, \neg\exists x_\alpha(h_{\circ\alpha} = (=_{\circ\alpha\alpha} x_\alpha)) \supset d_{\alpha(\circ\alpha)}(h_{\circ\alpha})\uparrow\}$.

Veamos un ejemplo para terminar. Asumamos que $\times_{\imath\imath\imath}, \div_{\imath\imath\imath} \in \mathcal{S}_c$. Sea Γ una axiomatización estándar de la aritmética sobre los racionales en la que $\times_{\imath\imath\imath}$ se define como producto pero $\div_{\imath\imath\imath}$ no ha sido definida aún. $\div_{\imath\imath\imath}$ puede ser definida en $\Gamma \cup \Delta$ como la división en términos de la siguiente ecuación: $B_\circ = [\div_{\imath\imath\imath}x_\imath y_\imath \sim d_{\imath(\circ\imath)}(\lambda z_\imath(x_\imath = \times_{\imath\imath\imath}y_\imath z_\imath))]$. Además, esta definición especifica que la división entre 0 es indefinida *i.e.* la siguiente expresión es verdadera: $\Gamma \cup \Delta \cup \{B_\circ\} \vdash \div_{\imath\imath\imath}x_\imath 0_\imath\uparrow$. Así pues, el operador de descripción definida $d_{\imath(\circ\imath)}$ proporciona los medios para definir la división a partir de la multiplicación de una forma natural [21].

2.4. Extensión LUTINS de PF

IMPS es un software creado por Farmer, Guttman y Thayer. IMPS fue programado principalmente en el lenguaje LISP y que funciona adecuadamente para el sistema operativo Linux [22]. El objetivo principal de IMPS es presentar un sistema de prueba matemática interactivo capaz de proveer de soporte mecánico a las técnicas tradicionales de razonamiento matemático. El sistema consiste en una base de datos matemáticos representados como una colección de teorías axiomáticas interconectadas y una colección de herramientas para explorar, aplicar y extender las

matemáticas. Esta forma de formalizar las matemáticas en una red de pequeñas teorías presenta varias ventajas para las matemáticas mecanizadas [23]. Las pruebas en IMPS son una combinación de computación e inferencia de alto nivel. En consecuencia, se parecen a pruebas informales inteligibles, pero a diferencia de las pruebas informales, todos los detalles de una prueba IMPS se verifican mecánicamente. La lógica de IMPS tiene como objetivo permitir al usuario formular conceptos y argumentos matemáticos de forma natural y directa. Es una teoría de tipos simple con un fuerte apoyo para especificar y razonar sobre funciones.

La lógica de IMPS es conocida con el nombre LUTINS por sus siglas en inglés: *Logic of Undefined Terms for Inference in a Natural Style*. LUTINS es esencialmente una extensión del sistema PF en la que se añaden algunos constructores de expresiones y constantes lógicas [24]. Gracias a estas actualizaciones en PF, LUTINS se convierte en una versión de la teoría de tipos simple en la que se admiten funciones parciales, términos que no denotan y también subtipos. Así pues, se escogió LUTINS como lógica de IMPS por su lenguaje familiar, su alta expresividad y la facilidad para tratar funciones parciales y funciones de orden superior. Esta elección llevó a IMPS al siguiente nivel, de hecho, muchos de los teoremas que se han probado con este software han alcanzado un nivel alrededor del teorema fundamental del cálculo [25].

En LUTINS los tipos de función son de la forma $\alpha_1 \times ... \times \alpha_n \to \alpha_{n+1}$. Un tipo es de clase $\circ$ si el tipo es $\circ$ o tiene la forma $\alpha_1 \times ... \times \alpha_n \to \alpha_{n+1}$ donde α_{n+1} es de clase $\circ$. El criterio para determinar la clase es el mismo que en PF. Uno de los defectos de PF es que los tipos indican el dominio y el rango de las funciones totales pero no el de las funciones parciales. Lo que se necesita para LUTINS es un sistema de subtipos y tipos para organizar las funciones parciales así como lo hace PF para las totales. De este modo, definiremos una especie como un subtipo o un tipo y crearemos un sistema de especies en PF para lograr LUTINS. Asimismo, el lenguaje de LUTINS contiene una jerarquía de especies atómicas y compuestas. A cada una de las especies atómicas se le asigna una especie adjunta que puede ser ella misma.

La relación entre la especie atómica y la adjunta se determina por un orden parcial $\preceq$ en las especies del lenguaje cumpliendo una serie de propiedades:

Si α es atómica y β adjunta, entonces $\alpha \preceq \beta$.
Si $\alpha_i \preceq \beta_i$ para cada i entre 1 y $n+1$, entonces $\alpha_1 \times ... \times \alpha_n \to \alpha_{n+1} \preceq \beta_1 \times ... \times \beta_n \to \beta_{n+1}$.
Los tipos de PF son las especies maximales en el orden parcial $\preceq$.
Para cada especie α hay un único tipo $\tau(\alpha)$ tal que $\alpha \preceq \tau(\alpha)$.

A modo de ejemplo, consideremos un lenguaje para los números reales que contiene las especies atómicas $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ y $\circ$ cuyos asjuntos son $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{R}$ y $\circ$, respectivamente. Entonces, $\mathbb{Z} \preceq \mathbb{Q} \preceq \mathbb{R}$ y $\mathbb{Q} \to \mathbb{Z} \preceq \mathbb{R} \to \mathbb{R}$. Claramente, siendo $\mathbb{Z}$, $\mathbb{Q}$ y $\mathbb{R}$ los enteros, los racionales y los reales respectivamente, tenemos que $\mathbb{Q} \to \mathbb{Z}$ denota el dominio de las funciones parciales de los racionales a los enteros; a su vez este dominio seria un subconjunto del dominio denotado por $\mathbb{R} \to \mathbb{R}$, este es, el dominio de las funciones parciales de los reales a los reales.

Los tipos se utilizan de tres maneras en LUTINS. En primer lugar, se utilizan para restringir los operadores vinculantes. Por ejemplo, el principio de Arquímedes de los números reales se expresa mediante $\forall x : \mathbb{R}.\exists y : \mathbb{Z}.x < y$. En segundo lugar, a cada expresión e se le asigna una especie $\sigma(e)$ en la base de su sintaxis. $\sigma(e) = \alpha$ significa que, si e es definido, el valor de e está en el conjunto denotado por α. Finalmente, las especies también facilitan la construcción de interpretaciones de una teoría de LUTINS en otra [25].

La calidad de "definido" en LUTINS se expresa mediante dos operadores. Asimismo, la expresión $e \downarrow \alpha$ debe leerse como "e es definido en α", lo que significa que e es definido y su valor es un miembro del conjunto denotado por α. La expresión $e \downarrow$, en cambio, se lee simplemente como "e es definido" y es una abreviatura de $e \downarrow \sigma(e)$. Puede resultar bastante útil mostrar un ejemplo de estos mecanismos. Asumamos que las especies $\mathbb{N}$ y $\mathbb{R}$ denotan los números naturales y los números reales y que la constante tg denota la función tangente. De este modo, $\mathbb{N} \preceq \mathbb{R}$ y $\sigma(tg) = \mathbb{R} \to \mathbb{R}$. Entonces $\sigma(tg(0)) = \sigma(tg(\pi/4)) = \sigma(tg(\pi/2)) = \mathbb{R}$ por la sintaxis de tg. Por otra parte, tenemos que $tg(0) \downarrow$, $tg(\pi/4) \downarrow \mathbb{N}$ y que $tg(\pi/2) \uparrow$, por la semántica de tg.

El defecto de LUTINS es que muchas preguntas sobre la definición de las expresiones deben responderse en el curso de una demostración. Por este motivo, es de vital importancia que cualquier sistema que implemente una lógica como LUTINS proporcione un soporte automatizado para verificar la definición de muchas expresiones, de lo contrario, los usuarios quedarían exhaustos al tener que demostrar una gran cantidad de teoremas, la mayoría de ellos triviales. Por este motivo, IMPS implementa un algoritmo de verificación de definición automatizado cuyas características pueden ser estudiadas en profundiad aquí (Farmer *et al.*, 1993) [23].

3. Discusión

En esta sección se exponen los argumentos a favor y en contra de cada uno de los sistemas lógicos presentados en el marco teórico. En primer lugar, se evaluan las diferencias de cada uno de los sistemas y en qué sentido son adecuados en términos matemáticos y de computación. Por este motivo, en segundo lugar se presenta un análisis pedagógico en el que se proponen algunas alternativas para fomentar la teoría de tipos dentro de algunos sectores en los que puede resultar beneficioso estudiarla.

3.1. Análisis lógico, matemático y computacional

La teoría de tipos de Church es un sistema formal en el que se asume que todas las funciones son totales y todos los términos tienen una denotación. Este sistema surge a raíz de algunas inquietudes matemáticas del siglo XX y en respuesta a la paradoja de Russell. Además, dió pie a algunos de los primeros lenguajes de programación. Sin embargo, algunas de las necesidades prácticas y teóricas en matemáticas e informática no quedan cubiertas por la teoría de Church. A pesar de los avances científicos, la motivación por la formalización de las matemáticas sigue vigente, así como las mejoras computacionales. Por este motivo han aparecido sistemas derivados de la teoría de tipos de Church que tratan de refinar la teoría de tipos o, al menos, aportarle un enfoque más natural.

Así pues, a raíz del sistema $\mathcal{Q}_0$ nacen los sistemas Alonzo, PF y LUTINS, los cuales tratan de representar de forma fiel algunos conceptos matemáticos como las funciones que no se aplican a todo el dominio, estas son, las funciones parciales.

Para lograr estos objetivos, Alonzo emplea una vía en la que se asigna un valor indefinido a las funciones como $\frac{1}{x}$, siendo $x = 0$. Este método requiere un menor número de cambios en la teoría de tipos inicial aportando una solución intuitiva en cuanto al trato de este tipo de funciones. No obstante, en la práctica matemática no se asocia ningún tipo de valor a estas funciones. En el caso de la computación, suele asociarse un *error* a estas funciones, el cual se propaga en los resultados sucesivos. Esto podría resultar beneficioso en términos de razonamiento informático pero no en términos de razonamiento matemático.

Por otra parte, el sistema PF emplea una aproximación a la representación de funciones parciales mediante la valuación total en fórmulas y la valuación parcial en términos. De este modo, funciones como $\frac{1}{x}$ no toman ningún valor cuando $x = 0$. De hecho, tampoco lo hacen cuando una función se anida a otras, es decir, las funciones $x + \frac{1}{x}$ o $\left(\frac{1}{x}\right)^2$ tampoco toman ningún valor, lo que representa de forma bastante fiel la forma en la que las matemáticas piensan acerca de

estas. A nivel computacional, es un sistema fácilmente implementable, de la misma forma que lo son $\mathcal{Q}_0$ o PF. Esto es debido a la propia estructura del universo de teoría de tipos. De hecho, en informática se emplea comúnmente la palabra "tipo" de una variable de un código para asociarle una categoria. Por ejemplo, "true" y "false" son de tipo *Bool* (Boolean), "gato" y "hola" son de tipo *string* (cadena) y los números 35 y 287 son de tipo *Int* (enteros). Asimismo, el propio código nos devuelve como resultado `<class 'int'>` si le escribimos `a=35` y `print(type(a))`[10]. Además, PF no se aproxima a las funciones parciales con la valuación *error*, lo que permite una mejor práctica matemática. Por lo tanto, aunque no refleja de la forma más fiel la visión de un computador, es un sistema con una implementación muy fácil y natural.

En el aspecto computacional, LUTINS no establece una mejora en cuanto a la concepción de las funciones parciales en informática. Sin embargo, su estructura de tipos y subtipos permite una optimización en los programas informáticos destinados a la prueba de teormas matemáticos. Como ya hemos visto, algunos *proof assistants*[11] como IMPS, imlementan LUTINS para su software. Esta complementación de la lógica con asistentes de prueba automatizados corrige los defectos de PF o LUTINS en cuanto a la amplia demostración de teoremas de la forma $A_\alpha \uparrow$ y $A_\alpha \downarrow$.

Atendiendo a los aspectos más formales, es interesante comparar $\mathcal{Q}_0$ y PF, pues además de ser las más conocidas, sus diferencias pueden ser mayores que con Alonzo o LUTINS, las cuales presentan solamente pequeñas modificaciones.

En el lenguaje formal, $\mathcal{Q}_0$ y PF difieren en asunciones sobre las constantes. En PF, el conjunto de las constantes es contable e incluye $=_{\circ\alpha\alpha}\ \forall\alpha \in \mathcal{T}$ mientras que en $\mathcal{Q}_0$ el conjunto puede ser incontable e incluye un operador $\imath_{\iota(\circ\iota)}\ \forall\alpha \in \mathcal{T}$[12]. (Farmer, 1990, p.1288) [21]. A nivel semántico, PF se basa en una extensión de la semántica de modelos generales de Henkin para que las funciones parciales y los términos no denotativos tengan cabida. De esta forma, un modelo general de $\mathcal{Q}_0$ lo es también para PF pero no a la inversa. De hecho, PF colapsa en $\mathcal{Q}_0$ si todas las funciones son totales. Ambas lógicas son igual de expresivas. Así pues, el tratamiento de la cuantificación es el mismo, así como el razonamiento proposicional, que también es el mismo. Otra de las similitudes es que ambas son completas, de hecho, sus pruebas de completitud son bastante similares, tal y como se ha comentado en el marco teórico. Una última similitud entre los sistemas de este trabajo es que solamente poseen una única regla de inferencia. Esta virtud compartida permite

[10] Este comando sirve para imprimir el tipo de la variable `a`.

[11] Los *proof assistants* son programas que corroboran que una prueba matemática sea correcta, algunos incluso son capaces de ayudar a hallar teoremas o demostrarlos. Suelen ser interactivos, con verificación automática y basados en un lenguaje formal.

[12] En la publicación de Farmer, se utiliza **H-A** para referirse a $\mathcal{Q}_0$.

que sea más fácil de implementar algorítmicamente ya que automatizar el proceso de inferencia o de verificación de prueba en un sistema con muchas reglas requiere un mayor esfuerzo y puede ser más propenso a errores. Por otra parte, con un mayor número de reglas de inferencia, existe un mayor riesgo de que la interpretación semántica de una afirmación se desvíe de su intención original.

En definitiva, se ha puesto de manifiesto que la teoría de tipos es un sistema elegante y simple sintácticamente, además de muy intuitivo y versátil ya que con ciertas modificaciones semánticas se consiguen sistemas de teoría de tipos apropiados para hablar de distintas áreas. Así pues, $\mathcal{Q}_0$ es un sistema adecuado para aquellos que quieran estudiar la lógica y la teoría de tipos de forma más abstracta y profunda, además de investigar en la formalización de las matemáticas o en las bases de la computación. PF, en cambio, puede ser útil para acercarse al razonamiento práctico matemático en teoría de tipos. Por lo tanto, es un sistema lógico que puede atraer a perfiles de distintos sectores. A los matemáticos puede atraerlos para corroborar demostraciones de teoremas con extensiones de PF y para la formalización de las matemáticas. A los científicos de datos también puede interesarles emplear lenguajes basados en teoría de tipos como Lisp para sus estudios. Incluso para los ingenieros resulta atractivo estudiar la teoría de tipos para modelizar y analizar algunos softwares. Esto nos deja, irremediablemente, en disposición de analizar de qué formas podemos incluir la teoría de tipos a nivel pedagógico para fomentar el desarrollo de otras disciplinas a través de la lógica.

3.2. Análisis pedagógico

La situación actual de la lógica en la filosofía y otras disciplinas ha sido la consecuencia de algunos factores políticos y académicos concretos [26]. Para Enrique Alonso, la lógica forma se trata simplemente del estudio de la capacidad expresiva y la potencia de cálculo (potencia de computación) de los lenguajes formales de la factura más variada. Así pues, la lógica encuentra sus contenidos en lo que se conoce como metalógica, muchas veces con la intención de alejarlos de las interpretaciones más escolares de la propia lógica contemporánea (E. Alonso, 2019, p.1-5) [26]. Además, Enrique Alonso afirma también que, en el ámbito español, la ausencia de este debate puede deberse a la escasa presencia de representantes en las sociedades que podrían acogerlo [26].

Así pues, debido a estas circunstancias académicas y políticas se considera oportuno un análisis pedagógico para la inclusión de la lógica en la enseñanza. Por lo tanto, en esta parte del trabajo, se discuten algunas de las vías posibles para incluir la teoría de tipos en el plan de estudios de aquellos sectores en los que puede ser beneficiosa. Concretamente, se proponen dos vías distintas: una estrategia que introduzca la lógica en etapas tempranas y una estrategia que incluya la teoría de tipos directamente en aquellas carreras donde sea de utilidad.

La primera estrategia es quizá la mas beneficiosa de las dos, básicamente porque bajo esta vía el alumnado obtendría los beneficios de la segunda vía ya que sería más fácilmente implementable, pero obtendría también otros beneficios que presenta esta vía. Sin embargo, esta primera vía requiere cambios estructurales y modificaciones en los planes de estudios de la mayoría de cursos académicos. Lo cual puede resultar costoso a nivel logístico y a nivel argumental ya que la lógica y la filosofia, desgraciadamente, suelen verse como disciplinas poco productivas en términos de desarrollo económico.

Esta primera estrategia pasa por la incorporación de la lógica en etapas muy tempranas, incluyendo juegos y problemas de lógica entretenidos y didácticos. Es decir, se incluyen actividades desde completar puzzles y secuencias sencillas hasta resolución de problemas proposicionales. Por ejemplo, los clásicos problemas en los que se debe adivinar quién ha sido el "ladrón" conociendo las características de cada uno de los candidatos. Así pues, en la etapa de la infancia se inciaría al alumnado con juegos y problemas para acercarlo paulatinamente a la lógica proposicional. De este modo, la lógica proposicional podría empezar a estudiarse en un sentido más formal aunque sencillo al acercarse la etapa adolescente. Finalmente, la lógica de primer orden podría verse inicialmente alrededor del bachillerato y, por lo tanto, la teoría de tipos se vería incorporada en los primeros años de grado, lo que ayudaría al alumnado a sentar unas bases de conocimiento, tanto para aquellos quienes quieran conocer los fundamentos matemáticos como para aquellos quienes quieran conocer la teoría subyacente a los lenguajes de programación. En definitiva, esta estrategia incentiva el estudio de los sistemas de teoría de tipos más abstractos com $\mathcal{Q}_0$ pero también a los más aplicados como LUTINS.

Esta estrategia es favorable para el alumnado también para evitar la frustración en las escuelas. La mayoría de niños y niñas afirman disfrutar de las matemáticas cuando las entienden. Pero comprender las matemáticas no significa otra cosa que seguir el razonamiento lógico subyacente, el cual les permite construir un concepto matemático abstracto o una estrategia de resolución (Antonia Huertas, 2000, p.2) [27]. Por este motivo, ya existen algunas propuestas para incluir la lógica en etapas tempranas y la lógica proposicional con un enfoque más formal a partir de los 12 años, entre ellas encontramos, por ejemplo, la de Antonia Huertas [27] o la de Olivier Gasquet, François Schwarzentruber y Martin Strecke [28]. Las etapas que se proponen en estos estudios pueden recordarnos rápidamente a la teoría del desarrollo cognitivo de Piaget. En la teoría del desarrollo cognitivo de Piaget se distinguen cuatro etapas: sensiomotora (hasta los 2 años), preoperacional (2-7 años), operacional concreta (7-11 años) y operacional formal (a partir de los 12 años) [29]. Entre estas etapas nos conciernen principlamente las tres últimas. En la preoperacional se

comienza a pensar de forma simbólica y pictórica, en la operacional concreta se comienza a pensar de forma lógica y en la operacional formal se adquiere la habilidad de pensar de forma hipotética y abstracta. Por lo tanto, vemos una clara correlación entre las etapas propuestas para cada ámbito de la lógica y estos modelos psicológicos. No obstante, existen diversas opiniones y algunos como Robert H. Ennis argumentan que en edades anteriores a los 12 años las personas podemos comprender la lógica proposicional, de hecho, tendría algunos beneficios presentarla a esta edad o antes [30].

A todo esto, se pueden poner en práctica también algunas propuestas pedagógicas relacionadas con el estudio de las situaciones didácticas. De esta forma, el profesorado debe preocuparse por los conocimientos previos del alumnado, sus ideas y experiencias para optimizar su rol como mediador del aprendizaje. Para ello, es importante evaluar también el contexto específico en el que se desarrolla la enseñanza y el aprendizaje. De hecho, crear escenarios ficticios relacionados con la experiencia cotidiana de los alumnos, permite que los estudiantes utilicen sus conocimientos previos, planteen hipótesis y generen nuevos aprendizajes significativos con la ayuda del profesor [31]. Esta forma de enseñar, creando una situación didáctica concreta, es fácilmente implementable para la lógica proposicional. Una vez más, podemos pensar en el problema de adivinar el "ladrón" pero en este caso con otro tipo de personajes que los propios alumnos puedan representar.

En la segunda estrategia, se propone incorporar la teoría de tipos directamente al ámbito universitario, concretamente en carreras como matemáticas, ingeniería informática u otras carreras afines. Esta estrategia es quizá algo más sencilla de conseguir ya que las disciplinas afectadas con cada vez más conscientes de los beneficios de la lógica y la teoría de tipos [32]. Además, los cambios propuestos no son cambios a gran escala. De hecho, en esta segunda vía se propone una combinación entre el estudio teórico y la aplicación práctica. Este sistema de enseñanza favorece principalmente a los sistemas de teoría de tipos más implementables como PF o LUTINS ya que los conocimientos sobre la lógica en el alumnado no serían tan sólido como en la primera estrategia. Sin embargo, existen algunas herramientas de razonamiento ético automático como LogiKEy, la cual se basa en la versión $\mathcal{Q}_0$ de teoría de tipos [33]. Esta herramienta se utiliza para mejorar la representación de conocimiento y la confiabilidad en inteligencia artificial, combinando el lenguaje expresivo de orden superior, la jerarquía de la teoría de tipos y la lógica deóntica [33].

Los ingenieros informáticos, específicamente los ingenieros de software emplean frecuentemente los asistentes de prueba. Como ya hemos visto, algunos de estos asistentes de prueba se fundamentan en sistemas de teoría de tipos y, algunos otros, se fundamentan en la teoría de categorías, la cual es un campo de la teoría de tipos para la lógica intuisionista donde se estudia el dominio y la programación de funciones [32]. Por este motivo, algunos como D. de Carvalho y N. Kudasov ya

proponen cursos de iniciación para los jóvenes programadores (D. de Carvalho y N. Kudasov, 2020, p.152) [32]. Además, como ya hemos discutido anteriormente, la teoría de tipos simples es una lógica de de orden superior que se alinea más estrechamente con la práctica matemática. Sus virtudes la hacen más natural, concisa y práctica. Por este motivo, no solamente Carvalho y Kudasov recomiendan enseñar la teoría de tipos simples en los cursos introductorios de lógica, sinó que Farmer también afirma que beneficia a los estudiantes, especialmente a los de ciencias de la computación e ingenierías [34]. Esto es debido a que la teoría de tipos proporciona habilidades prácticas de lógica para aplicaciones del mundo real.

Aunque pueda parecer contradictorio, en ocasiones los estudiantes comprenden mejor la teoría habiendo resuelto anteriormente situaciones prácticas. Por ejemplo, a lo largo de mi etapa en el grado de física, he conocido varios estudiantes que reolvían primero los problemas planteados para después entender la teoría. Esta es una forma de aprender que, aunque *prima facie* contraproducente, efectiva. Algunas disciplinas como la medicina implementan este tipo de enseñanza práctica [35]. Además, algunos estudios miden el impacto del aprendizaje experiencial [36].

La conjunción de todos estos factores puede contribuir al éxito de esta segunda estrategia pedagógica, logrando así que los ingenieros y científicos puedan mejorar la verificación de software, el desarrollo de algoritmos, las bases de datos, los avances en inteligencia artificial y el estudio de los fundamentos de la teoría de la computación. Por otra parte, esta segunda vía puede incorporarse en los másteres universitarios, lo que permitiría que estudiantes con un perfil tan interdisciplinar como los filósofos tuvieran un acceso menos restringido a estas disciplinas. Además, esta segunda vía se presenta en etapas más maduras y en sectores donde se promueve la aplicación de la teoría. Estos aspectos son también esperanzadores para aquellos que deseen formarse en remoto o de forma semipresencial. De hecho, la Universitat Oberta de Catalunya tiene algunos cursos en línea sobre lógica destinados a estudiantes de ciencias de la computación [37].

4. Conclusiones

Una de las principales motivaciones del trabajo era conocer la forma en la que surge la teoría de tipos, cómo evoluciona y a qué innovaciones abre las puertas. El estudio del contexto histórico ha permitido cumplir estas aspiraciones. Por lo tanto, hemos podido ver que la teoría de tipos surge en un periodo de ebullición para las matemáticas y la lógica. Concretamente, surge con el afán de lidiar con la paradoja de Russell. Hemos podido ver como esta solución se basa en un universo que no es plano, es decir, que establece una jerarquía de tipos. En el contexto, hemos podido ver también cómo el interés por la teoría de tipos se volvió algo general, especialmente para aquellos que veían la teoría de tipos como una alternativa a la formalización de las matemáticas. Posteriormente, los avances en computación se aliaron con la teoría de tipos para crear nuevos sistemas, lenguajes y tecnologías. Así pues, el interés en computación y la formalización de las matemáticas conllevó la formación de nuevos sistemas de teoría de tipos.

De esta forma, el trabajo nos ha llevado a una segunda sección en la que se presentan formalmente algunos de estos sistemas. Los sistemas presentados se basan en la teoría de tipos de Church. El sistema $\mathcal{Q}_0$ es la versión de Andrews para la teoría de tipos de Church, en la que se presenta una notación más sencilla. Por otra parte, hemos visto también otros sistemas como Alonzo, PF o LUTINS, los cuales surgen para dar cuenta de las funciones parciales de una forma más natural, tal y como vemos en las matemáticas aplicadas. Esta segunda sección cumple también con el objetivo de comprender en un sentido más formal y profundo la teoría de tipos. De este modo, no solo nos hemos interesado en cómo surgen los sistemas que preceden o suceden a la teoría de tipos, sinó que hemos atendido también a los fundamentos de cada uno de ellos. Asimismo, se ha presentado un esquema de la sintaxis, la semántica y los principios de estos sistemas para establecer una comparativa entre ellos a distintos niveles.

Seguidamente, se ha realizado un análisis comparativo entre los sistemas basado en la teoría expuesta en el marco teórico. A nivel lógico, todos los sistemas son elegantes y con una sintaxis clara y muy parecida, de hecho, solamente algunas abreviaturas son las que permiten ser más o menos laxos con la notación. Evidentemente, debido a las diferencias semánticas y la aparición de términos no denotativos, la sintaxis se adapta a estos principios de forma natural. Dichos principios son los que separan a unos sistemas de otros, ya que son los que permiten la aparición de funciones parciales y términos no denotativos y, en el caso de LUTINS, una jerarquía de tipos y subtipos. En general, son muchas las similitudes entre las lógicas ya que todas son completas e igual de expresivas. Además unas colapsan en las otras con pequeñas modificaciones. Esto demuestra también que con algunos cambios y sin alterar demasiado la lógica, se pueden lograr sistemas de teoría de tipos más adecuados a nuestros objetivos, ya sean la formalización de las matemáticas con

un razonamiento más natural o bien la implementación de la teoría de tipos en computación. En cuanto a la implementación, se ha podido corroborar la hipótesis de que la teoría de tipos es muy adecuada a nivel informático debido a su estructura y su única regla de inferencia.. Además, sistemas como LUTINS son el lenguaje de softwares como IMPS, un asistente de prueba automatizado.

Una vez puestos en evidencia los beneficios de la teoría de tipos en la lógica, las matemáticas y la computación, se ha procedido a comprobar la viabilidad de las hipótesis pedagógicas, las cuales persiguen el objetivo de incorporar la teoría de tipos en beneficio del desarrollo tecnológico y académico del alumnado. En primer lugar, se ha optado por una vía estructural en la que los planes de estudios incluyen la lógica de distintas formas y en distintos niveles. Bajo esta estrategia, los estudiantes solucionan problemas lógicos desde edades muy tempranas de forma adaptada a sus capacidades. A nivel formal, la lógica proposicional se incluye también en cursos mucho menos tardíos. No obstante, esto no significa que los estudiantes vayan a fracasar por una dificultad añadida académicamente. En realidad, como ya hemos visto, muchos estudios e investigadores respaldan estos modelos pedagógicos para la lógica, algunos incluso en etapas anteriores. Esta vía, además de beneficios cognitivos, tiene beneficios en cuanto al desarrollo lineal de la lógica académicamente. Por lo tanto, da pie a implementar la segunda vía, es decir, el aprendizaje práctico, de una forma más sencilla y con un conocimiento más sólido de la teoría. La segunda vía, se centra principalmente en el aprendizaje práctico. Por este motivo, fomenta el estudio de la teoría de tipos en los primeros cursos de grados en los que se traten aspectos computacionales. Empezar a aprender la teoría de tipos con sus aplicaciones prácticas puede parecer algo intempestivo, sin embargo, son muchos los estudiantes que asimilan mejor los conceptos teóricos cuando se les presentan casos prácticos de sus disciplinas. Posteriormente, acabarían incorporando la teoría en sus esquemas de razonamiento, lo que les permitiría crear códigos de programación más robustos. Otro de los beneficios de esta vía es la facilidad de implementación respecto a la primera. Esto se debe a que no requiere de tantos cambios. Además puede ser más sencillo convencer sobre las ventajas de la teoría de tipos a los implicados ya que muchos cursos ya empiezan a incorporarla.

En definitiva, podemos concluir que los objetivos del trabajo se han alcanzado de forma satisfactoria ya que se ha evualuado cómo y porqué surge la teoría de tipos y cuáles son sus frutos. Además, se han analizado en profundidad algunos de los sistemas, lo cual ha sido de gran ayuda para establecer una comparativa entre ellos y decidir cuáles se adaptan mejor a los objetivos computacionales y cuáles se adaptan mejor a la formalización de las matemáticas. Concretamente, se ha corroborado que los sistemas de teoría de tipos con funciones parciales son una gran alternativa para estas disciplinas. Por este motivo, se han expuesto dos vías para incorporar la teoría de tipos con funciones parciales a los planes de estudios.

Referencias

[1] Jean van Heijenoort. *From Frege to Gödel*. Harvard University Press, 1967.

[2] Traducción de Peter Long y Roger White. *Gottlob Frege: Philosophical and mathematical correspondence*. Oxford, Blackwell: 1980, p.130-170.

[3] Stanford Encyclopedia of Philosophy. *Type Theory*. Publicada por primera vez el miércoles 8 de febrero de 2006; revisión sustantiva el martes 6 de septiembre de 2022. `https://plato.stanford.edu/entries/type-theory/#TypeTheoTheo`

[4] David W. Roach, David A.Bednar. *The theory of logical types: A tool for understanding Levels and Types of Change in Organizations*. Human Relations, Vol.50, N.6: 1997.

[5] Stanford Encyclopedia of Philosophy. *Russell's Paradox*. Publicada por primera vez el viernes 8 de diciembre de 1995; revisión sustantiva el lunes 12 de octubre de 2020. `https://plato.stanford.edu/entries/russell-paradox/`

[6] Mary Falk de Losada. *Corrientes de Pensamiento Matemático del siglo XX*. Revista Científica General José María Córdova: 2012, p.2-6.

[7] Kurt Gödel. *On the completeness of the calculus of logic*: 1929.

[8] Kurt Gödel. *On Formally Undecidable Propositions of Principia Mathematica and Related Systems*: 1930.

[9] Alonzo Church. *A Formulation of the Simple Theory of Types*. The Journal of Symbolic Logic, Vol.5, N.2: 1940, p.56-68.

[10] María Manzano. *La piedra filosofal*. Universidad de Salamanca: 2007.

[11] William Farmer. *5a Simple Type theory*. Clase virual en McMaster University: 2020. `https://www.youtube.com/watch?v=OSt7bMxscJ4&t=425s`

[12] Lawrence C. Paulson y Christoph Benzmüller. *LEO II: An Effective Higher-Order Theorem Prover*. University of Cambridge: 2007. `https://www.cl.cam.ac.uk/~lp15/Grants/LEO-II/index.html`

[13] María Manzano. *Extensions of First Order Logic*. Universidad de Barcelona: 1996, p148-180.

[14] Alan Turing. *On computable numbers, with an application to the Entscheidungsproblem*: 1936.

[15] María Manzano. *VII Escuela de Invierno del Master en Lógica y Filosofía de la Ciencia. Día Mundial de la Lógica: Gottlob Frege*. Ponencia en la Universidad de Salamanca en el Día Mundial de la Lógica 2020 y VII Escuela de Invierno del Máster en Lógica y Filosofía de la Ciencia. `https://www.youtube.com/watch?v=nlAQaPVA6Dc&t=796s`

[16] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Applied Logic Series, Vol.27, Carnegie Mellon University, Pittsburgh, Pennsylvania, U.S.A: 2002.

[17] Stanford Encyclopedia of Philosophy. *Church's Type Theory*. Publicada por primera vez el viernes 25 de agosto de 2006; revisión sustantiva el jueves 16 de enero de 2024. https://plato.stanford.edu/entries/type-theory-church/#ForBasEqu

[18] William farmer. *Simple Type Theory: A Practical Logic for Expressing and Reasoning About Mathematical Ideas*. McMaster University, Hamilton, Canada: 2023.

[19] William Farmer. *5b Simple Type theory*. Clase virual en McMaster University: 2020. https://www.youtube.com/watch?v=yd2UG43DFe4&t=486s

[20] William Farmer. *5d Simple Type theory*. Clase virual en McMaster University: 2020. https://www.youtube.com/watch?v=0A-DNWijJso&t=1046s

[21] William M. Farmer. *A Partial Functions Version of Church's Simple Theory of Types*. The Journal of Symbolic Logic, Vol.55, N.3: 1990.

[22] W. M. Farmer, J. D. Guttman, F. J. Thayer. *An Interactive Mathematical Proof System*. The MITRE Corporation: 1990-2006. https://imps.mcmaster.ca/#:~:text=IMPS%20is%20an%20Interactive%20Mathematical%20Proof%20System%20intended,to%20be%20partial%20and%20terms%20to%20be%20undefined.

[23] W. M. Farmer, J. D. Guttman, F. J. Thayer. *IMPS: An Interactive Mathematical Proof System*. The MITRE Corporation, 202 Burlington Rd, Bedford, U.S.A: 1993.

[24] W. M. Farmer. *A simple type theory with partial functions and subtypes*. The MITRE Corporation, 202 Burlington Rd, Bedford, U.S.A: 1993.

[25] William M. Farmer. *Reasoning about Partial Functions with the Aid of a Computer*. Kluwer Academic Publishers, Erkenntnis: 1995.

[26] Enrique Alonso. *Lógica y Teoría de la argumentación: anatomía de una reforma*. Universidad Autónoma de Madrid, España: 2019.

[27] Antonia Huertas. *First International Congress on Tools for Teaching Logic Teaching Logical reasoning in high school*. Barcelona: 2000.

[28] Olivier Gasquet, François Schwarzentruber y Martin Strecker. *The Computational Power of Propositional Logic Shown to Beginners*. IRIT (Institut de Recherche en Informatique de Toulouse): 2011.

[29] Smaragda Kazi y Evangelia Galanaki. *Piagetian Theory of Cognitive Development.* Encyclopedia of Child and Adolescent Development: 2020.

[30] Robert H. Ennis. *Children's Ability to Handle Piaget's Propositional Logic: A Conceptual Critique.* Review of Educational Research Vol.45, N.1, p. 1-41. American Educational Research Association: 1975.

[31] Guy Brousseau. *Iniciación al estudio de la teoría de las situaciones didácticas.* Buenos Aires: Libros del Zorzal, 2007.

[32] Daniel de Carvalho y Nikolai Kudasov. *Teaching Logic, from a Conceptual Viewpoint.* Springer Nature Switzerland AG, J.-M. Bruel *et al.* (Eds.): FISEE 2019, LNCS 12271, pp. 151–177, 2020.

[33] Christoph Benzmüller, Xavier Parent, Leendert van der Torre. *Designing Normative Theories for Ethical and Legal Reasoning: LogiKEy Framework, Methodology, and Tool Support*: 2020.

[34] William M. Farmer. *The seven virtues of simple type theory.* Ontario, Canadá: 2008.

[35] E. González-López, I. García-Lázaro, A. Blanco-Alfonso, A. Otero-Puime. *Aprendizaje basado en la resolución de problemas: una experiencia práctica.* Educación Médica versión impresa ISSN 1575-1813 Educ. méd. Vol.13 N.1: 2010.

[36] Estrella Magdalena Espinar Álava y José Alberto Vigueras Moreno. *El aprendizaje experiencial y su impacto en la educación actual.* Rev. Cubana Edu. Superior Vol.39 N.3 La Habana: 2020.

[37] Antonia Huertas, Josep M. Humet, Laura López y Enric Mor. *The SELL Project: A Learning Tool for E-Learning Logic.* Computer Science Department, Universitat Oberta de Catalunya: 2011.