Universidad de Valladolid

La indecidibilidad de la lógica de primer orden y el método de árboles

TRABAJO FIN DE MÁSTER

Máster Interuniversitario en Lógica y Filosofía de la Ciencia

AUTORA: Alba Lavagnoli González

TUTOR: Juan Barba Escribá

La indecidibilidad de la lógica de primer orden y el método de árboles

Resumen

El teorema de Church demuestra que la cuestión de la validez o no de fórmulas de primer orden no es resoluble por máquinas de Turing ni procedimientos equivalentes, lo que a partir de la tesis de Church-Turing significa la inexistencia de métodos efectivos de decisión para determinar si una fórmula de primer orden es válida o no.

El método deductivo de árboles para la lógica proposicional sí que es un método de decisión para esta lógica, porque el árbol siempre se termina tanto si se cierran todas las ramas del árbol como si no. Por el contrario, en lógica de primer orden el método de árboles no es un método efectivo, puesto que en algunos casos no se acaban nunca. A tenor de lo dicho más arriba, tampoco podría nunca serlo, aun cuando intentásemos complementarlo de alguna manera. A partir de ahí intentaremos desmenuzar las razones que hacen que los árboles de primer orden no sean ni puedan ser un método de decisión y las implicaciones que eso tiene.

Palabras clave: máquinas de Turing, *halting problem*, método de decisión, método deductivo de árboles.

INDICE

Introducción

La importancia de los teoremas de incompletitud de Gödel es evidente a la hora de enfocar el presente trabajo. Los teoremas de Gödel vienen a decir, respectivamente, que en todo sistema formal para la aritmética existen proposiciones formalmente indecidibles y que es imposible encontrar una prueba de consistencia de un sistema, suponiendo que tal sistema sea consistente, dentro del propio sistema. Pero hay que ir más allá de esto, ya que los teoremas de incompletitud de Gödel supusieron la debacle del programa formalista de Hilbert.

No olvidemos que Hilbert se propuso fundamentar las matemáticas, al igual que hizo con los fundamentos de la geometría, pero se dio cuenta de la imposibilidad de tal propósito, por lo cual se limitó a presentar de manera sistemática los problemas que consideró más cruciales.

> "Who of us would not be glad to lift the veil behind which the future lies hidden; to cast a glance at the next advances of our science and at the secrets of its development during future centuries? What particular goals will there be toward which the leading mathematical spirits of coming generations will strive? What new methods and new facts in the wide and rich field of mathematical thought will the new centuries disclose?" (Hilbert, 1900).

Nuestro interés y objetivo del trabajo se dirigirá a uno de estos grandes problemas planteados por Hilbert: el problema de la decisión, o *Entscheidungsproblem*, cuya solución no puede discernirse de la identificación de un sistema para definir un procedimiento de cálculo dado de manera algorítmica o mecánica.

Por tanto, dicho trabajo tendrá como punto de partida estos dos pilares que se anudan en lo que se conoce como la crisis de los fundamentos de las matemáticas, y es justo donde Alan Turing empezará a moverse y teorizar, echando las bases de lo que será la teoría de la computabilidad.

Con sus resultados sobre la incompletitud, Gödel no acabó simplemente con el sueño hilbertiano de poder tratar a todas las matemáticas como un sistema formal; sino que, con la definición de las funciones primitivas recursivas, el descubrimiento de proposiciones formalmente indecidibles y la imposibilidad de demostrar la consistencia de un sistema con un procedimiento mecánico, el trabajo de Gödel fue un fuerte estímulo para obtener una definición precisa de la noción de algoritmo.

A partir de los teoremas de incompletitud y del problema sin resolver de Hilbert, el *Entscheidungsproblem*, que podríamos formular del siguiente modo: ¿existe algún método por el cual se pudiera decidir, para cualquier proposición matemática dada, si dicha proposición es o no demostrable en una teoría axiomática dada?, vemos como la principal dificultad de esta pregunta radica en dar una definición formal y general de lo que se entiende por expresiones como "método" o "procedimiento efectivo".

El presente trabajo constará de tres partes. En el primer punto analizaremos la necesidad de una caracterización precisa del concepto de algoritmo para poder responder al problema de la decisión. En el siglo XX, el concepto de algoritmo se formalizó para resolver el problema matemático de la decisión (*Entscheidungsproblem*), establecido por David Hilbert en 1928, y otras formalizaciones posteriores llegaron con el desarrollo de los conceptos de "calculabilidad efectiva" y "método efectivo". Las formalizaciones matemáticas más famosas son las funciones recursivas de Gödel, de Herbrand y de Kleene; el cálculo lambda de Church; y, finalmente, las máquinas de Turing. A pesar de esto, todavía falta una definición del concepto de algoritmo que sea formal y no técnica y, por lo tanto, nos vemos obligados a seguir aceptando y usando una idea intuitiva de lo que es un algoritmo, por ejemplo:

"una secuencia ordenada y finita de pasos elementales (operaciones o instrucciones) que conduce a un resultado definido en un tiempo finito".

De estos resultados, entre otras cosas, surgió la tesis de Church-Turing, que establece que cualquier algoritmo se puede calcular con una máquina de Turing. En otras palabras, esta tesis argumenta que es esencialmente imposible tratar de imaginar un modelo de algoritmo más poderoso y, en consecuencia, que ninguna máquina puede resolver problemas que una máquina de Turing no pueda resolver en principio. Este no es un teorema probado matemáticamente, ya que la tesis establece la igualdad de dos conceptos, el de algoritmo y el de la máquina de Turing, pero el primero no tiene una definición formal. Actualmente, la tesis es aceptada y compartida.

Se expondrá como dicho concepto, el de algoritmo, está enlazado con el concepto de máquina de Turing, que sin "querer" es él mismo un algoritmo. Turing introdujo estas máquinas en el contexto de la investigación sobre los fundamentos de las matemáticas. Concretamente, utilizó estos dispositivos abstractos para demostrar que no existe un método o procedimiento general efectivo para resolver, calcular o computar

cada instancia del *Entscheidungsproblem*. Además, presentaremos otros conceptos que han resultado ser equivalentes tanto al concepto de máquina de Turing como al mismo concepto de algoritmo. Esta equivalencia es lo que se conoce como la Tesis de Church-Turing, que será determinante para el desarrollo de dicho trabajo. Y el último punto de este primer apartado estará dedicado a explicar los problemas indecidibles que surgen a raíz de la respuesta negativa al problema de la decisión. Un ejemplo de problema indecidible será el problema de la parada y como Turing lo detectó.

El problema de la parada guarda una estrecha relación con la demostración de la indecidibilidad de la lógica de primer orden, que es lo que tratará el segundo apartado del presente trabajo. En éste explicaremos detenidamente dicho problema de indecidibilidad demostrándolo por medio de máquinas de Turing, mostrando que un método de decisión para la validez de fórmulas de la lógica de primer orden conllevaría un método de decisión para el problema de la parada, cosa que, como hemos dicho, Turing demostró que es imposible. Y que nos lleva directamente al último apartado del trabajo, el método de deducción por árboles.

En este último apartado presentaremos el funcionamiento del método de árboles tanto en el caso de la lógica proposicional como el caso de la lógica de primer orden, para detenernos en los árboles infinitos que se dan en la lógica de primer orden, en los cuales es fundamental la comprensión de las reglas de los cuantificadores. El caso de los árboles infinitos nos llevará a presentar las similitudes que guardan éstos con el problema de la parada y con la demostración de la indecidibilidad de la lógica de primer orden. Veremos que la analogía entre árboles infinitos y máquinas que no se paran es muy estrecha, con lo que mostramos que, incluso desde un punto de vista puramente intuitivo, es patente que el problema de la parada y el de la decisión en lógica de primer orden son esencialmente el mismo.

1. Teoría de la computabilidad

1.1 La noción de algoritmo y el *Entscheidungsproblem*

En general, un algoritmo es un método para resolver un problema. Un algoritmo consiste en un conjunto finito de instrucciones que se pueden aplicar de forma determinista; además, la solución, si existe, debe poder obtenerse en un número finito de pasos. Por lo tanto, los algoritmos (también llamados métodos efectivos) son procedimientos que permiten resolver un problema de una manera completamente mecánica, sin recurrir a ninguna forma de creatividad o intuición.
Vinculada a la noción de algoritmo tenemos la noción de función efectivamente computable. Se dice que una función (en el sentido matemático del término) es calculable (o computable) de una manera algorítmica, o incluso efectivamente calculable (computable), syss hay un algoritmo que, para cada argumento, calcula su valor.

Por tanto, un proceso de cálculo consiste en la idea de identificar un método para obtener un cierto resultado, la cual ya se había manifestado en los orígenes del pensamiento científico, y la descripción de cómo resolver un problema matemático tiene raíces igualmente antiguas. En el siglo XIX, las matemáticas experimentaron un desarrollo notable, y una especialización posible gracias a la aplicación sistemática de modelos, que la hizo ser percibida como la ciencia exacta por definición e indujo una formalización creciente basada en estrictas reglas lógicas, algoritmos y en el método axiomático deductivo: a partir de conceptos asumidos como conocidos a priori, a través de reglas bien definidas llegamos a la demostración de otras proposiciones.

Hasta la mayor parte del siglo XIX, los algoritmos, como el de Euclides o Fibonacci, parecían satisfacibles, ya que permitían replicar los procedimientos descritos. Pero a principios del siglo XX, los matemáticos comenzaron a pensar que estos procedimientos eran solo una aproximación que no deberíamos dar por satisfechos. Russell y Whitehead publicaron los *Principia Mathematica*, en el que se afirmaba la posibilidad de que todas las matemáticas podrían tratarse con reglas formales lógicas. Años más tarde, Hilbert y Ackermann lanzaron el desafío del *Entscheidungsproblem*, es decir, plantearon el problema de la decisión: ¿dada cualquier declaración de la lógica de primer orden –que son las proposiciones generalmente utilizadas en matemáticas- existe un procedimiento capaz de establecer si es demostrable o refutable, es decir, si en realidad es derivable de un conjunto dado de proposiciones que se supone que son verdaderas? En

otras palabras, se trataba de identificar un algoritmo que, al menos en teoría, siempre conduciría a un resultado.

La pregunta fue de gran interés, ya que las diferentes ramas de las matemáticas pueden presentarse a través de sistemas axiomáticos, de los cuales se derivan los teoremas a través de pasos lógicos. Resolver el problema de la decisión habría permitido a los matemáticos concentrarse en la formulación de axiomas, dejando la tarea de demostración a un algoritmo.

La necesidad de definir, para cada teoría matemática formalizada, cuáles son las características que debe tener para considerarse consistente y completa, se convirtió en una de las principales urgencias: desde principios de siglo XX, el descubrimiento de antinomias había planteado el problema en términos precisos.

Turing hizo algunas contribuciones fundamentales a esta teoría naciente. En 1935 conoció el *Entscheidungsproblem* y decidió estudiarlo, obteniendo una respuesta negativa al problema que planteaba. Al mismo tiempo, de una manera completamente independiente, el mismo resultado negativo también fue obtenido por Alonzo Church.

Sin embargo, en comparación con el enfoque de Church, Turing siguió un camino diferente y original: para analizar el concepto de proceso algorítmico, definió una clase de máquinas abstractas – las que hoy en día se conocen como "máquinas de Turing"-. El enfoque de Turing tiene la ventaja de abordar el problema de manera extremadamente directa, sin presuponer nociones o herramientas matemáticas preexistentes. Además, las máquinas de Turing desempeñarán más tarde un papel importante en el estudio del cálculo automático, ya que pueden interpretarse como modelos idealizados de las calculadoras digitales actuales. Cada algoritmo se presta, al menos en principio, para ser automatizado, es decir, para ser ejecutado por una máquina adecuadamente diseñada.

1.2 Las máquinas de Turing y la Tesis de Church-Turing

1.2.1 Las máquinas de Turing

Intuitivamente, tenemos un algoritmo (o un método de decisión) para resolver un problema syss tenemos una lista finita de instrucciones tales que:

1. A partir de los datos iniciales, las instrucciones se pueden aplicar de una manera estrictamente determinista, es decir, que en cada paso siempre es posible establecer de forma inequívoca cuál es la instrucción que se debe aplicar en el siguiente paso;

2. Existe un criterio único para determinar cuándo se ha alcanzado un estado final, es decir, cuándo el proceso debe considerarse terminado y el resultado, si lo hubiera, se ha obtenido;
3. Un estado final siempre debe ser alcanzable en un número finito de pasos. Llamaremos *input* a los datos de inicio del cálculo; y *output*, su resultado. Se dice que una función es calculable de forma algorítmica (o calculable de manera efectiva, o realmente calculable) si existe un algoritmo que permite calcular los valores para todos los argumentos.

Turing introdujo las máquinas de Turing en el contexto de la investigación sobre los fundamentos de las matemáticas. Concretamente, utilizó estos dispositivos abstractos para demostrar que no existe un método o procedimiento general efectivo para resolver, calcular o computar cada instancia del *Entscheidungsproblem*, a saber, el problema para decidir para cada enunciado en la lógica de primer orden si es decidible o no en esa lógica.

Para abordar este problema, se necesita una noción formal de "procedimiento efectivo" y las máquinas de Turing estaban destinadas a hacer exactamente eso. Para ello, Turing se enfrentó al problema de proporcionar un equivalente riguroso del concepto intuitivo de algoritmo, definiendo un modelo de la actividad de un ser humano que está realizando un cálculo algorítmico. De hecho, en el artículo *On computable numbers with an application to the Entscheidungsproblem* Turing propone su definición de "máquina de computación"[1]:

> "We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions q_1,q_2, …, q_R which will be called "m-configurations". The machine is supplied with a "tape", (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the *r*-th, bearing the symbol $\mathfrak{S}(r)$ which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned symbol". The "scanned symbol" is the only one of which the machine is, so to speak, "directly aware". However, by altering its *m*-configuration the machine can effectively remember some of the symbols which it has "seen" (scanned) previously. The possible behaviour of the machine at any moment is determined by the *m*-configuration q_n and the scanned symbol $\mathfrak{S}(r)$. This pair q_n, $\mathfrak{S}(r)$ will be called the "configuration": thus the configuration determines the possible behaviour of the machine." (Turing 1936-7: 231)

[1] Las "máquinas automáticas" de Turing, como las denominó en 1936, fueron diseñadas específicamente para la computación de números reales. Pero fueron nombradas "máquinas de Turing" por Alonzo Church en su revisión del artículo de Turing (Church 1937).

Las instrucciones de las MdT unas veces se presentan como quíntuplas y otras como cuádruplas[2]. En ambos casos, los dos primeros elementos son el estado *q* en que está la máquina y el símbolo *s* que está escrutando. Los otros dos o tres indican lo que hay que hacer. En la versión de quíntuplas, o sea, la versión de Turing, los tres restantes dicen

1. el símbolo que hay que escribir;
2. hacia dónde hay que moverse, si a la izquierda o a la derecha;
3. el nuevo estado.

En la versión de cuádruplas uno de los dos elementos indica o bien el símbolo que se escribe (y la máquina no se mueve de casilla) o bien hacia dónde se mueve (y en este caso no hay cambio de símbolo). En ambos caso el cuarto elemento de la instrucción indica el cambio de estado interno. Lo que en la versión de quíntuplas se hace en un paso, en la de cuádruplas se hace en dos, pero en todo caso da igual presentar las máquinas de una u otra formas: con más o menos pasos, se hacen exactamente las mismas cosas[3].

Claramente esto no influye en la formalización de una máquina de Turing, que podemos definir del siguiente modo:

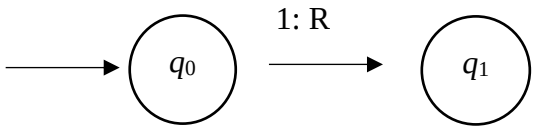
$M = (Q, \Gamma, \Sigma, \delta, \Delta, q_0, F)$, donde

- Q es conjunto finito de estados, $q_0, q_1, \ldots, q_i$.
- Σ es conjunto finito de símbolos, el "alfabeto de la máquina".
- Γ es conjunto finito de símbolos, el "alfabeto de la cinta". Por definición, $\Sigma \subseteq \Gamma$.
- δ es la función de transición. Indica el movimiento de la máquina.
  $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$
- Δ es el símbolo para indicar la casilla vacía. Por tanto, $\Delta \in \Gamma \wedge \Delta \notin \Sigma$
- q_0 es el estado inicial de la máquina. Por tanto, $q_0 \in Q$.
- F es un subconjunto finito de los estados finales, o configuraciones finales de la máquina. Por tanto, $F \subseteq Q$.

[2] La versión original, es decir, la propuesta por Turing, está en quíntuplas, pero Emil Post las estandarizó en cuádruplas. Es importante apuntar esto, para que no haya ningún problema de entendimiento, porque en el presente trabajo, usaremos las MdT con instrucciones cuádruplas.

[3] Volvemos a señalar la decisión de usar instrucciones cuádruplas porque también la demostración del teorema de Church es a base de cuádruplas.

Por tanto, si mandamos ejecutar a la maquina la siguiente instrucción cuádrupla q_0 1 R q_1



La máquina en estado q_0 ejecuta la instrucción y se desplaza a la derecha, R, escribiendo el símbolo 1 en la nueva casilla. La instrucción se da por terminada con la máquina que asume un nuevo estado, q_1.

Dado que cada casilla puede contener un único símbolo, al escribir uno nuevo, el símbolo presente en dicha casilla debe considerarse borrado. Al realizar un cálculo, un ser humano tiene en cuenta las operaciones ya realizadas y los símbolos escaneados previamente a través de su memoria, es decir, cambiando su "estado mental". Para simular esto, supondremos que una máquina puede asumir, dependiendo de los eventos anteriores del proceso de cálculo, un cierto número de estados internos (uno y no más de uno a la vez), que corresponden a los "estados mentales" del ser humano. Dichos estados serán finitos en número, ya que "if we admitted an infinity states of mind, some of them will be 'arbitrarily close' and will be confused" (Turing 1936-7: 250). Sin embargo, limitarse a un número finito de estados no constituye un límite, ya que "the use of more complicated states of mind can be avoided by writing more symbols on the tape" (Turing 1936-7: 250).

Además, es importante tener en cuenta que la existencia de al menos una configuración final es una condición necesaria, pero no suficiente, para que el cálculo de una MdT termine. De hecho, es fácil construir una MdT que, aunque en su tabla de instrucciones aparezca una o más configuraciones finales, de lugar a un cálculo que avanza hasta el infinito para algunos *inputs* posibles.

1.2.2 La Tesis de Church-Turing

Como se ha explicado, las máquinas de Turing originalmente estaban destinadas a formalizar la noción de computabilidad para abordar un problema fundamental de las matemáticas. Independientemente del trabajo de Turing, Alonzo Church dio una formulación diferente pero lógicamente equivalente. Hoy en día, la mayoría de la comunidad científica-informática está de acuerdo en que la noción formal de Turing captura *todos* los problemas computables, a saber: para cualquier problema computable, hay una máquina de Turing que lo computa. Esto se conoce como la *tesis de Church-Turing*. Implica que, si se acepta, cualquier problema no computable por una máquina de Turing no es computable por ningún medio finito. De hecho, dado que la ambición de

Turing era capturar "[all] the possible processes which can be carried out in computing a number." (Turing 1936: 249), se deduce que, si aceptamos el análisis de Turing:

- Cualquier problema no computable por una máquina de Turing no es "computable" en sentido absoluto (al menos, absoluto relativo a los humanos).
- Para cualquier problema que creamos que sea computable, deberíamos poder construir una máquina de Turing que lo calcule.

Por lo cual, la tesis de Church-Turing se refiere al concepto de un método *eficaz* o *sistemático* o *mecánico* en la lógica, las matemáticas y la informática. Un método, o procedimiento, *M*, para lograr el resultado deseado se llama "eficaz"" (o "sistemático" o "mecánico") syss:

1. *M* se establece en términos de un número finito de instrucciones exactas (cada instrucción se expresa por medio de un número finito de símbolos);
2. *M*, si se lleva a cabo sin errores, producirá el resultado deseado en un número finito de pasos;
3. *M* puede (en la práctica o en principio) ser llevada a cabo por un ser humano sin ayuda de ninguna maquinaria, excepto el papel y el lápiz;
4. *M* no exige ninguna visión, intuición o ingenio por parte del ser humano que lleva a cabo el método.

Turing demostró que, dada su tesis, no puede haber *un* método eficaz en el caso del cálculo completo de primer orden: ninguna máquina de Turing puede decir, de cada fórmula del cálculo de primer orden, si la fórmula es o no un teorema del cálculo (siempre que la máquina se limite a un número finito de pasos al probar una fórmula para la teorema).

Tesis de Turing: That LCMs (Logical Computing Machines) can do anything that could be described as 'rule of thumb' or 'purely mechanical'. (Turing, 1948)

Pero, ¿por qué se conoce como Tesis de Church-Turing?

Church también se interesó en este problema hilbertiano y declaró lo siguiente: "By the Entscheidungsproblem of a system of symbolic logic is here understood the problem to

find an effective method by which, given any expression Q in the notation of the system, it can be determined whether or not Q is probable in the system" (Church, 1936b: 41). Unos meses antes del trabajo de Turing, Church llegó al mismo resultado negativo. Descubrieron este resultado de forma bastante independiente el uno del otro. Church empleó el concepto de *definibilidad lambda* donde Turing utilizaba la computabilidad por un L.C.M. Se dice que una función es definida por lambda si los valores de la función se pueden obtener mediante un determinado proceso de sustitución repetida.

Cuando Turing habla de lo «puramente mecánico», Church habla de lo «efectivamente calculable» para indicar que existe un método eficaz para obtener los valores de la función. Además, Church propuso: "define the notion, […] of an *effectively calculable* function of positive integers by identifying it with the notion of a recursive function of positive integers". (Church, 1936ª: 356)

Después de enterarse de la propuesta de Church de 1936 para identificar la efectividad con la definibilidad lambda (mientras preparaba su propio documento para su publicación) Turing estableció rápidamente que el concepto de definibilidad lambda y su concepto de computabilidad son equivalentes, demostrando que "the theorem that all effectively calculable sequences are computable and its converse" (Turing, 1936: 263). Por lo tanto, en la propuesta de Church, las palabras "función definible de enteros positivos" (e igualmente las palabras "función recursiva de enteros positivos") pueden ser sustituidas por "función de enteros positivos que es computable por la máquina Turing".

Esta es, pues, la "hipótesis de trabajo" que, en efecto, Church propuso:

> **Tesis de Church**: Una función de enteros positivos es efectivamente calculable sólo si es lambda-definible (o, equivalentemente, recursivo).[4]

Aunque se podría reformular del siguiente modo:

> **Tesis general de Church**: Toda función efectivamente calculable es recursiva general.

[4] La implicación inversa, que cada función lambda-definible de enteros positivos es efectivamente calculable, se conoce comúnmente como *la inversa de la tesis de Church*, aunque Church no hizo tal distinción, ya que agrupó ambas tesis en la definición que propone.

Si la atención se limita a las funciones de los enteros positivos, la tesis de Church y la tesis de Turing son *extensionalmente* equivalentes, es decir, que calculan sobre una misma clase de funciones, aunque la manera de aplicación sea diferente, pero ambas calculan sobre lo mismo: la clase de funciones definidas por lambda (de enteros positivos) es idéntica a la clase de funciones recursivas (de enteros positivos) y a la clase de funciones computables (de enteros positivos). Téngase en cuenta, que si bien las dos tesis son equivalentes en este sentido, sin embargo tienen significados distintos y por lo tanto son dos tesis *diferentes*[5].

Pero al ser equivalentes es lo que lleva a que se conozca como *Tesis de Church-Turing*.

> **Tesis de Church-Turing:** *Cualquier cálculo realizado de manera efectiva es computable.*

En su repaso de la obra de Turing, Church mismo reconoció la superioridad del análisis de la eficacia de Turing: "computability by a Turing machine […] has the advantage of making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately" (Church 1937: 43).

Si bien se ha puesto numerosas veces en tela de juicio la validez de la tesis de Church-Turing, el resumen de la situación que Turing proporcionó en un artículo de 1948 sigue siendo cierto incluso hoy en día: "It is found in practice that L.C.M.s can do anything that could be described as'rule of thumb' or 'purely mechanical'. This is sufficiently well established that it is now agreed amongst logicians that 'calculable by means of an L.C.M.' is the correct accurate rendering of such phrases" (Turing, 1948: 414).

1.2.2.1 Razones para aceptar la Tesis de Church-Turing

Podemos proporcionar tres razones suficientes para aceptar la Tesis de Church-Turing:

- (1) Todas las funciones efectivamente calculables que han sido investigadas en este sentido han resultado ser computables por la máquina Turing.

[5] Una diferencia importante entre las dos es que la tesis de Turing se refiere a *las máquinas de computación,* mientras que la de Church no.

- (2) Todos los métodos u operaciones conocidos para obtener nuevas funciones efectivamente calculables a partir de funciones efectivamente calculables se ajustan paralelamente a los métodos para la construcción de nuevas máquinas Turing a partir de máquinas Turing dadas.
- (3) Todos los intentos de dar un análisis exacto de la noción intuitiva de una función efectivamente calculable han resultado ser *equivalentes,* en el sentido de que cada análisis ofrecido ha demostrado caracterizar la misma clase de funciones, a saber, las que son computable por la máquina Turing.

Estas razones son similares a las que Kleene enuncia para validar la Tesis de Church, que como sabemos es equivalente a la de Turing. Kleene proporciona tres evidencias, a saber: (A) la evidencia heurística; (B) la equivalencia de distintas formulaciones, y (C) el concepto de máquina computable proporcionado por Turing. (Kleene, 1952)

1.3 Problemas indecidibles

El análisis del concepto de algoritmo propuesto por Turing tiene importantes consecuencias desde el punto de vista teórico. En particular, se desprende de la existencia de problemas indecidibles, o sea, problemas que no pueden ser resueltos por ningún algoritmo.

En particular, Turing demostró que no es efectivamente decidible el problema de la parada (*halting problem*) para las MdT. El problema de la parada pregunta si siempre es posible, descrito un algoritmo y un determinado *input* finito, establecer si el algoritmo en cuestión finaliza o continúa la ejecución sin parar hasta el infinito.

Por lo tanto, el *halting problem* consiste en establecer si, dada una MdT *M* y una entrada *n*, *M* con *input n* da lugar a un cálculo que termina o no. A esto le sigue una respuesta negativa para el *Entscheidungsproblem* de Hilbert: el problema de la parada puede ser formulado como un problema aritmético; sea *L* un sistema lógico que formaliza la aritmética (por ejemplo, un sistema formal basado en axiomas de Peano formalizados en la lógica de primer orden), se demuestra que *L* no puede ser decidible, porque si lo fuera, el problema de la parada también debería ser decidible.

La indecidibilidad del problema para una MdT significa que no existe un algoritmo que, dada una MdT genérica y dado un *input* genérico para ello, siempre permite establecer si el cálculo de esa máquina con esa entrada termina o no.

La demostración de la indecidibilidad del problema de la parada para las MdT tiene muchas consecuencias a nivel teórico. De ello se desprende la indecidibilidad de otros problemas, y está estrechamente relacionada con los resultados limitantes de la lógica matemática, incluidos los teoremas de la incompletitud de Gödel.
En informática, el resultado significa que hay problemas que no pueden, incluso en principio, ser resueltos por una computadora; por ejemplo, no puede haber un programa que siempre pueda establecer si un programa escrito en un lenguaje determinado con una determinada entrada terminará o no terminará su cálculo.

1.3.1 Demostración del problema de la parada

Supongamos que existe una máquina *M'* tal que aplicada a una máquina *M* y un input *n*, *M'* (*M*,*n*) devuelve 1 si *M* se para cuando se aplica al input *n* y devuelve 0 si *M* no se para al aplicarse al input *n*. En particular, *M'* (*M*,*n*) devuelve 1 syss *M*(*n*) se para.

Podemos asumir que las máquinas computan con números, así que cuando decimos que *M'* se aplica a *M* y *n* en rigor se aplica a dos números, el número de *M* (representémosle con [*M*]) y a *n*, así que lo que tenemos es (1) *M'* ([*M*],*n*)=1 syss *M*(*n*) se para.

Entonces podemos construir una segunda máquina *M''* añadiendo a *M'* instrucciones que se ejecutarían antes y después del procesamiento de *M'* de tal manera que

- Primero se añaden instrucciones que duplican el input dado.
- Después se ejecutan las instrucciones de *M'*.
- Finalmente, se añaden instrucciones tales que si la ejecución de las instrucciones de *M'* acaba en 1, las instrucciones adicionales llevan a *M''* a un bucle, de forma que donde *M'* da 1, *M''* no se para.

Tomemos ahora una máquina cualquiera *M*, y apliquemos *M''* al input [*M*].

- Lo primero que hará *M''* es duplicar el input [*M*], produciendo el par de números idénticos [*M*],[*M*].
- Después se ejecutarán las instrucciones de *M''* sobre el par de números [*M*],[*M*] obteniendo como resultado 1 si *M*([*M*]) se para y 0 si no se para.
- Finalmente, si el resultado anterior era 1, *M''* se meterá en bucle y si era 0 se parará.
- Así pues, para cualquier máquina *M* tenemos que

(*) $M([M])$ se para syss $M'([M],[M])=1$ syss $M''([M])$ no se para.

Ahora, veamos que pasa si aplicamos M'' a su propio número, $[M'']$. De acuerdo con (*), tendremos $M''([M''])$ se para syss $M''([M''])$ no se para. Pero eso es una contradicción, así que no puede existir tal máquina M''. Puesto que, si existiera M' sería fácil construir M'', la conclusión es que M' no puede existir tampoco.

No existe una máquina de Turing capaz de resolver el problema de la parada y asumiendo la Tesis de Church-Turing, no existe ningún método efectivo para ello.

2. La indecidibilidad de la lógica de primer orden

2.1 Demostración de la indecidibilidad de la lógica de primer orden

2.1.1 La formalización de las máquinas de Turing en un lenguaje de primer orden

Basándonos en la demostración y en el resultado de la indecidibilidad del problema de la parada, veremos cómo es indecidible también el problema de decisión de la lógica de primer orden, es decir, queremos demostrar que no hay una máquina de Turing que puede decidir si una determinada fórmula es o no un teorema de la lógica de primer orden[6]. Dada una MdT *M* y un input *i* cualesquiera, podemos construir una fórmula de primer orden F_{Mi} que es válida syss *M* se para cuando se aplica al input *i*.

Recordemos que:

- Una fórmula válida es una fórmula que siempre es verdad, es decir, en al caso de primer orden, que es satisfecha por todos los modelos y asignaciones.
- Una fórmula A es deducible de un conjunto de fórmulas Γ (en un sistema deductivo dado) en símbolos $\Gamma \vdash A$, si usando las reglas del sistema se puede deducir A a partir de Γ. Γ puede ser un conjunto de fórmulas finito o infinito (y, como caso particular de conjunto finito, puede ser el conjunto vacío).
- Una fórmula A es consecuencia de un conjunto de fórmulas Γ, en símbolos $\Gamma \vDash A$, si no existe ninguna interpretación (es decir, un modelo y una asignación) que satisfaga todas las fórmulas de Γ pero no satisfaga A. Como antes, Γ puede ser un conjunto de fórmulas finito (vacío o no).

Además, por el teorema de la deducción sabemos que cuando Γ es un conjunto finito de fórmulas, el hecho de que A sea deducible, o consecuencia, de Γ equivale a que la fórmula $\wedge\Gamma \rightarrow A$ es una fórmula válida, donde $\wedge\Gamma$ representa la conjunción de las fórmulas de Γ (en un ejemplo, $B,C,D \vDash A$ equivale a que $B \wedge C \wedge D \rightarrow A$ es una fórmula válida. Por eso, todos los casos de consecuencia (y de deducibilidad) con conjuntos de premisas finitos se reducen a validez de fórmulas. Si el conjunto de premisas es infinito entonces no existe la conjunción de todas las premisas y no se puede aplicar el teorema de la deducción.

[6] Seguiremos las notaciones y la demostración propuesta en [Boolos, 2007]

De hecho el teorema de indecidibilidad de Church dice que el conjunto de fórmulas válidas de primer orden es indecidible, esto es, que no existe ningún algoritmo (o ninguna máquina de Turing, si aceptamos la tesis Church-Turing) que permita decidir si una fórmula de primer orden es válida o no. Por eso, la decisión sobre consecuencia/deducibilidad con finitas premisas finitas equivale a decisión sobre validez de fórmulas, así que se podría decir que el teorema de Church establece la indecidibilidad de la deducibilidad/consecuencia de primer orden con finitas premisas.

Por tanto, para demostrar este resultado, mostraremos que dada cualquier máquina de Turing *M* y cualquier número natural *n* podemos describir un conjunto finito de fórmulas de primer orden Γ y una fórmula *A* tal que Γ implica *A* syss la máquina *M* se detiene cuando encuentra el número *n* en la cinta. La idea base que nos hará de guía en la búsqueda del conjunto finito de fórmulas Γ y la fórmula *A* será que las fórmulas en Γ describirán en las instrucciones de la máquina *M* y el input *n* dado[7].
Entonces, si fuera posible resolver el problema de la decisión de la consecuencia lógica, es decir, si fuera posible decidir si la fórmula *A* es deducible de las fórmulas en Γ, también podríamos decidir sobre el problema de la parada. Como vimos en los apartados 1.2.2 y 1.3, por la tesis de Church-Turing, el problema de la parada no es decidible, por lo cual, siempre por la tesis de Church-Turing, el problema de decisión de la lógica de primer orden no es decidible, de lo que se sigue que la lógica de primer orden sea indecidible.

Como en el caso de las máquinas de Turing tenemos que empezar introduciendo simultáneamente el lenguaje en el que las fórmulas en Γ y la fórmula *A* estarán descritas y su *interpretación estándar* $\mathcal{M}$. El lenguaje de interpretación dependerá de qué máquina y qué entrada *n* estemos considerando. El dominio de $\mathcal{M}$ será en todos los casos el de los números enteros. Se usarán enteros no negativos para numerar los instantes en que la máquina está funcionando: la máquina inicia en el momento 0. Los enteros también se usarán para numerar las casillas en la cinta: la máquina comienza en la casilla 0, y las casillas a la izquierda y a la derecha son numeradas como sigue.

[7] Pero, sobre todo, hay que señalar que intuitivamente $\Gamma \vdash A$ significa que de la aplicación de las instrucciones al input *n* se deduce que la máquina se para. Además, por lo dicho más arriba, $\Gamma \vdash A$ equivale a que la fórmula $\wedge \Gamma \rightarrow A$ es válida. Por lo tanto, si la validez de primer orden fuera decidible, podríamos saber si $\wedge \Gamma \rightarrow A$ es válida o no y con ello, si *M* aplicada a *n* se para o no, lo que resolvería el problema de la parada.

…	-3	-2	-1	0	1	2	3	…

Habrá una constante **0**, cuya denotación en la interpretación estándar será ser cero, y predicados binarios **S** y <, cuyas denotaciones serán la relación de ser sucesor (la relación que un entero *n* tiene con *n* + 1 y nada más) y la relación de orden de los enteros, respectivamente. En cuanto a los otros predicados, tendremos para cada uno de los estados, que no son de parada, de la máquina, numerados, digamos, desde 1 (el estado inicial) hasta *k*, un predicado unario. En la interpretación estándar, Q_i denotará el conjunto de $t \geq 0$ tal que en el momento numerado *t* la máquina está en el estado numerado *i*. A parte de esto necesitamos otros dos predicados binarios **@** y **M**. La denotación del primero será el conjunto de pares de enteros $t \geq 0$ y *x*, de modo que en el tiempo numerado *t*, la máquina esté en la casilla numerada *x*. Y la denotación del último predicado será el conjunto de $t \geq 0$ y *x* de tal manera que en el tiempo *t*, la casilla *x* está "marcada", es decir, contiene un símbolo en lugar de estar vacío[8]. Formalmente, la función de una variable se señala por su posición en el primer o segundo lugar del predicado **@** o **M**).

A continuación, debemos describir las fórmulas que se van a introducir en Γ y la fórmula *A*. Las fórmulas en Γ se agruparán en tres grupos. El primero contiene cierta "información básica" sobre **S** y < que será la misma para cualquier máquina y cualquier *input*. El segundo grupo consiste en una sola sentencia específica que describe el *input n* que estamos considerando. Y el tercero y último grupo consiste en una fórmula para cada instrucción "normal" de la máquina específica que estamos considerando, es decir, para cada instrucción excepto aquellas que nos dicen cuando se pararía la máquina.

La "información básica" está formada por las siguientes fórmulas:

(1) $\forall u \forall v \forall w(((\mathbf{S}uv \wedge \mathbf{S}uw) \rightarrow v = w) \wedge ((\mathbf{S}vu \wedge \mathbf{S}wu) \rightarrow v = w))$

(2) $\forall u \forall v(\mathbf{S}uv \rightarrow u < v) \wedge \forall u \forall v \forall w((u < v \wedge v < w) \rightarrow u < w)$

(3) $\forall u \forall v(u < v \rightarrow u \neq v)$

Estas fórmulas nos dicen que un número tiene solo un sucesor y solo un predecesor, que un número es menor que su predecesor, y así sucesivamente, y todos son igualmente verdaderos en la interpretación estándar.

Es oportuno, también, introducir abreviaturas para la relación de *m*-sucesor

[8] Usamos *t* como la variable para indicar el tiempo, instancia, o momento en el que se encuentra la máquina, y con *x* e *y* la numeración de las casillas.

$\mathbf{S}_0uv$ para $u = v$

$\mathbf{S}_1uv$ para $\mathbf{S}uv$

$\mathbf{S}_2uv$ para $\exists y(\mathbf{S}uy \wedge \mathbf{S}yv)$

$\mathbf{S}_3uv$ para $\exists y_1 \exists y_2(\mathbf{S}uy_1 \wedge \mathbf{S}y_1\, y_2 \wedge \mathbf{S}y_2v)$

y así sucesivamente. (En $\mathbf{S}_2$, y puede ser cualquier variable distinta de u y v; del mismo modo para $\mathbf{S}_3$.) Las siguientes fórmulas son verdaderas en la interpretación estándar.

(4) $\forall u \forall v \forall w(((\mathbf{S}_m uv \wedge \mathbf{S}_m uw) \rightarrow v = w) \wedge ((\mathbf{S}_m vu \wedge \mathbf{S}_m wu) \rightarrow v = w))$

(5) $\forall u \forall v(\mathbf{S}_m uv \rightarrow u < v)$ si $m \neq 0$

(6) $\forall u \forall v(\mathbf{S}_m uv \rightarrow u \neq v)$ si $m \neq 0$

(7) $\forall u \forall v \forall w(((\mathbf{S}_m wu \wedge \mathbf{S}uv) \rightarrow \mathbf{S}_k wv)$ si $k = m + 1$

(8) $\forall u \forall v \forall w(((\mathbf{S}_k wv \wedge \mathbf{S}uv) \rightarrow \mathbf{S}_m wu)$ si $m = k - 1$

De hecho, estas son consecuencias lógicas de (1) – (3) y, por tanto también de Γ, y son verdaderas en cualquier interpretación donde Γ es verdadero: (4) se sigue de la aplicación repetida de (1); (5) se sigue de la aplicación repetida de (2); (6) se sigue de (3) y (5); (7) es inmediato a partir de las definiciones; y (8) se sigue de (7) y (1).

Pero todavía necesitamos algunas notaciones adicionales antes de escribir el resto de fórmulas de Γ.

Aunque oficialmente nuestro lenguaje contiene solo el número 0 y no los números 1, 2, 3 o −1, −2, −3, es conveniente escribir $y = \mathbf{1}$, $y = \mathbf{2}$, $y = \mathbf{-1}$, y para $\mathbf{S}_1(\mathbf{0}, y)$, $\mathbf{S}_2(\mathbf{0}, y)$, $\mathbf{S}_{-1}(\mathbf{0}, y)$, y así sucesivamente. Razonando un poco podemos mostrar que con las convenciones anteriores (6)-(8) (aplicando el **0** a w) nos daría lo siguientes en donde **p**, **q**, y sucesivos, representan las expresiones genéricas del metalenguaje para los números p, q, y así sucesivamente:

(9) $\mathbf{p} \neq \mathbf{q}$ si $p \neq q$

(10) $\forall v(\mathbf{Sm}v \rightarrow v = \mathbf{k})$ si $k = m + 1$

(11) $\forall u(\mathbf{S}u\mathbf{k} \rightarrow u = \mathbf{m})$ si $m = k - 1$

Estas abreviaturas nos permiten escribir las restantes fórmulas de Γ.

El único miembro de Γ perteneciente al *input n* es la configuración inicial en el tiempo 0:

(12) $\mathbf{Q_0 0} \wedge \mathbf{@00} \wedge \mathbf{M00} \wedge \mathbf{M01} \wedge \ldots \wedge \mathbf{M0n} \wedge$
$\forall x(x \neq \mathbf{0} \wedge x \neq \mathbf{1} \wedge \ldots \wedge x \neq \mathbf{n} - \mathbf{1}) \rightarrow \neg \mathbf{M0}x)$

Esta es verdadera en la interpretación estándar, ya que en el instante 0 la máquina está en el estado 1, en la casilla 0, con las casillas desde 0 hasta *n* marcados para representar el *input n*, y todos las demás casillas están vacías.

Para completar Γ, habrá una fórmula para cada instrucción de no-parada, o sea, para cada instrucción de la siguiente forma, en donde *j* no es el estado de parada:

(*) Si está en el estado *i* y está escaneando el símbolo *e*,
entonces —— y pasa al estado *j*.

Al escribir la fórmula correspondiente de Γ, usamos una notación adicional, a veces escribiendo $\mathbf{M}$ como $\mathbf{M}_1$ y $\neg\mathbf{M}$ como $\mathbf{M}_0$. Por lo cual $\mathbf{M}_e tx$ significa, en la interpretación estándar, que en el tiempo *t*, la casilla *x* contiene el símbolo *e* (donde $e = 0$ significa el espacio en blanco, y $e = 1$ significa el símbolo marcado). Entonces la fórmula correspondiente a (*) tendrá la siguiente forma

(13) $$\forall t\forall x((\mathbf{Q}_i t \wedge \mathbf{@}tx \wedge \mathbf{M}_e tx) \rightarrow \exists u(\mathbf{S}tu \wedge \text{——} \wedge \mathrm{Q}_j u \wedge \forall y((y \neq x \wedge \mathbf{M}_1 ty) \rightarrow \mathbf{M}_1 uy) \wedge \forall y((y \neq x \wedge \mathbf{M}_0 ty) \rightarrow \mathbf{M}_0 uy)))$$

Las dos últimas sentencias dicen que la marca de casillas distintas de *x* permanece sin cambios desde un tiempo *t* al próximo tiempo *u*.

Lo que estaría en el espacio '——' en (13) depende de lo que esté en el correspondiente espacio en (*). Si la instrucción es (permanecer en la misma casilla *x* pero) imprime el símbolo *d*, la conjunción que falta en (9) será

$$\mathbf{@}ux \wedge \mathbf{M}_d ux$$

Si la instrucción es moverse una casilla a la derecha o a la izquierda (dejando el símbolo de la casilla *x* como era), sería

$$\exists y(\mathbf{S}_{\pm 1} xy \wedge \mathbf{@}uy \wedge (\mathbf{M}ux \leftrightarrow \mathbf{M}tx))$$

(con el signo menos para la izquierda y el signo más para la derecha). Si completásemos la fórmula de esta manera, (13) corresponde exactamente a la instrucción (*), y será verdadera en la interpretación estándar.

Esto completaría el conjunto Γ. Por tanto, lo siguiente será describir la fórmula *A* y para obtener dicha fórmula, consideremos una instrucción de parada, es decir, una instrucción intuitiva del siguiente tipo

(†) Si está en el estado *i* y está escaneando el símbolo *e*, debe pararse.

Para cada una de esas instrucciones, necesitaríamos escribir la siguiente fórmula

(14) $$\exists t \exists x(\mathbf{Q}_i t \wedge \mathbf{@}tx \wedge \mathbf{M}_e tx)$$

Esta será verdadera en la interpretación estándar syss en el curso de sus operaciones la máquina finalmente llega a una configuración donde la instrucción a aplicar es (†) y se detiene por esta razón. Sea *A* la disyunción de todas las fórmulas que tengan como forma (14) para todas las instrucciones de parada (†). Dado que la máquina finalmente se detendrá syss finalmente llega a una configuración donde la instrucción aplicable es alguna instrucción de parada u otra, la máquina eventualmente se detendrá syss *A* es verdadera en la interpretación estándar.

A partir de esto, queremos mostrar que Γ implica *A* syss la máquina dada, que comenzó con el *input* dado, finalmente se detiene. La parte "solo si" es fácil de demostrar. Todas las fórmulas en Γ son verdaderas en la interpretación estándar, mientras que *A* es verdadera solo si la máquina dada comenzó con el *input* dado finalmente se detiene. Si la máquina no se detiene, tenemos una interpretación donde todas las fórmulas en Γ son verdaderas y *A* no lo es, entonces Γ no implica *A*.

Para la parte "si" necesitamos añadir una noción más a nuestro lenguaje. Si $a \geq 0$ es un instante de tiempo en el que la máquina no se ha detenido (todavía), nos referimos con *descripción del tiempo* a la fórmula que hace para *a* lo qué (12) hace para 0, diciéndonos en qué estado se encuentra la máquina, dónde está y qué casillas están marcadas en el instante *a*. En otras palabras, si en algún instante *a* la máquina está en el estado *i*, en la casilla *p*, y las casillas marcadas son $q_1, q_2, \ldots, q_m$, entonces la descripción del tiempo *a* es la siguiente fórmula:

(15) $$\mathbf{Q}_i\mathbf{a} \wedge \mathbf{@ap} \wedge \mathbf{Maq}_1 \wedge \mathbf{Maq}_2 \wedge \ldots \wedge \mathbf{Maq}_m \wedge$$
$$\forall x((x \neq \mathbf{q}_1 \wedge x \neq \mathbf{q}_2 \wedge \ldots \wedge x \neq \mathbf{q}_m) \rightarrow \neg\mathbf{Ma}x)$$

Es importante tener en cuenta que (15) proporciona, directa o indirectamente, la información de que la máquina está escaneando un espacio en blanco o un símbolo en el instante *a*. Si la máquina está escaneando un símbolo, entonces *p* es uno de los q_r para $1 \leq r \leq m$, y $\mathbf{M}_1\mathbf{ap}$, que es **Map**, en realidad es el conjunto de los elementos de la conjunción de (15). Si la máquina está escaneando un espacio en blanco, entonces *p* es diferente de cada uno de los diversos números *q*. En este caso $\mathbf{M}_0\mathbf{ap}$, es decir ¬**Map**, está implícito en

(15) y en Γ. En pocas palabras, la razón es que (9) da $\mathbf{p} \neq \mathbf{q}_r$ para cada q_r, y la última conjunción de (15) proporciona **¬Map**[9].

Si la máquina se detiene en el momento $b = a + 1$, eso significa que en el instante a teníamos una configuración para la cual la instrucción aplicable sobre qué hacer a continuación era una instrucción de parada de tipo (†). En ese caso, $\mathbf{Q}_i\mathbf{a}$ y **@ap** serán conjunciones de la descripción del tiempo a, y $\mathbf{M}_e\mathbf{ap}$ también será una conjunción de la descripción (si $e = 1$) o una implicación lógica de la descripción y de Γ (si $e = 0$). De modo que (14) y, por lo tanto, A será un implicación lógica de Γ junto con la descripción del tiempo a. Pero, ¿qué pasa si la máquina no se detiene en el tiempo $b = a + 1$?

Lema. Si $a \geq 0$ y $b = a + 1$ es un instante en el que la máquina no se ha (todavía) parado, entonces Γ y la descripción del tiempo a implican la descripción del tiempo b.

Demostración: la prueba es ligeramente diferente para cada uno de los cuatro tipos de instrucciones de la máquina de Turing (imprimir un espacio en blanco, imprimir un símbolo, moverse hacia la izquierda o hacia la derecha); así que nos detendremos en la instrucción de imprimir un símbolo, que a su vez se subdivide en el caso inusual donde ya hay un símbolo en la casilla escaneada, de modo que la instrucción es solo para cambiar el estado, y el caso más habitual donde la casilla escaneada está en blanco. Consideramos solo este último caso.

Entonces la descripción del tiempo a sería la siguiente:

(16) $$\mathbf{Q}_i\mathbf{a} \wedge \mathbf{@ap} \wedge \mathbf{Maq}_1 \wedge \mathbf{Maq}_2 \wedge \ldots \wedge \mathbf{Maq}_m \wedge \forall x((x \neq \mathbf{q}_1 \wedge x \neq \mathbf{q}_2 \wedge \ldots \wedge x \neq \mathbf{q}_m) \rightarrow \neg\mathbf{Ma}x)$$

donde $p \neq q_r$ para cualquier r, entonces Γ implica $\mathbf{p} \neq \mathbf{q}_r$ por (9) y, por el argumento dado anteriormente, Γ y (16) juntos implican **¬Map**. La fórmula en Γ correspondiente a la instrucción aplicable sería la siguiente:

---

[9] Dicho de manera menos breve pero más precisa, lo que abrevia el adjunto de (15) sería

$$\forall x((\neg\mathbf{S}_{q1}\mathbf{0}x \wedge \ldots \neg\mathbf{S}_{qm}\mathbf{0}x) \rightarrow \neg\exists t(\mathbf{S}_a\mathbf{0}t \wedge \mathbf{M}tx)).$$

Y (9) se aplica a las abreviaturas de p y q_r es

$$\neg\exists x((\mathbf{S}_p x \wedge \mathbf{S}q_r x).$$

Estos juntos implican

$$\neg\exists t\exists x(\mathbf{S0}t \wedge \mathbf{S0}x \wedge \mathbf{M}tx)$$

que equivale a lo que abrevia ¬**Map**.

(17) $\forall t \forall x((\mathbf{Q}_i t \wedge @tx \wedge \neg \mathbf{M}tx) \rightarrow$
$\exists u(\mathbf{S}tu \wedge @ux \wedge \mathbf{M}ux \wedge \mathrm{Q}_j u \wedge \forall y((y \neq x \wedge \mathbf{M}ty) \rightarrow \mathbf{M}uy)$
$\wedge \forall y((y \neq x \wedge \neg \mathbf{M}ty) \rightarrow \neg \mathbf{M}uy)))$

Mientras que, la descripción del tiempo *b* sería la siguiente:

(18) $\mathbf{Q}_j\mathbf{b} \wedge @\mathbf{bp} \wedge \mathbf{Mbp} \wedge \mathbf{Mbq}_1 \wedge \mathbf{Mbq}_2 \wedge \ldots \wedge \mathbf{Mbq}_m \wedge$
$\forall x((x \neq \mathbf{p} \wedge x \neq \mathbf{q}_1 \wedge x \neq \mathbf{q}_2 \wedge \ldots \wedge x \neq \mathbf{q}_m) \rightarrow \neg \mathbf{Mb}x)$

Con lo cual (18) es una consecuencia de (16), (17) y Γ[10].

Ahora bien, la descripción del tiempo 0 es una de las fórmulas en Γ. Por el lema anterior, si la máquina no se detiene en el tiempo 1, la descripción del tiempo 1 será una consecuencia de Γ, y si la máquina no se detiene en el momento 2, la descripción del tiempo 2 será consecuencia de Γ junto con la descripción del tiempo 1, y así sucesivamente hasta el último instante *a* antes de que la máquina se pare, si alguna vez lo hace. Si se detiene en el momento $a + 1$, hemos visto que la descripción de tiempo a, que ahora sabemos que es una consecuencia de Γ, implica *A*. Por lo tanto, si la máquina se detiene, Γ implica *A*.

Hemos establecido que si el problema de decisión para la implicación lógica tuviera solución, el problema de la parada también la tendría, pero, por la tesis de Church-Turing, sabemos que no lo es. Por lo tanto, siempre por la tesis de Church-Turing el problema de decisión para la implicación lógica de primer orden es insoluble (Teorema de Church).

La lógica de primer orden es indecidible.

---

[10] Sustituyendo *a* por *t* y *p* por *x* en (17), obtenemos

$(\mathbf{Q}_i\mathbf{a} \wedge @\mathbf{ap} \wedge \neg \mathbf{Map}) \rightarrow$
$\exists u(\mathbf{Sa}u \wedge @u\mathbf{p} \wedge \mathbf{M}u\mathbf{p} \wedge \mathrm{Q}_j u \wedge$
$\forall y((y \neq \mathbf{p} \wedge \mathbf{Ma}y) \rightarrow \mathbf{M}uy) \wedge \forall y((y \neq \mathbf{p} \wedge \neg \mathbf{Ma}y) \rightarrow \neg \mathbf{M}uy))$

Por (16) y Γ implica $(\mathbf{Q}_i\mathbf{a} \wedge @\mathbf{ap} \wedge \neg \mathbf{Map})$ y obtenemos

$\exists u(\mathbf{Sa}u \wedge @u\mathbf{p} \wedge \mathbf{M}u\mathbf{p} \wedge \mathrm{Q}_j u \wedge$
$\forall y((y \neq \mathbf{p} \wedge \mathbf{Ma}y) \rightarrow \mathbf{M}uy) \wedge \forall y((y \neq \mathbf{p} \wedge \neg \mathbf{Ma}y) \rightarrow \neg \mathbf{M}uy))$

Y por (10), $\mathbf{Sa}u$ es $u = \mathbf{b}$, donde $b = a + 1$, obteniendo

$@\mathbf{bp} \wedge \mathbf{Mbp} \wedge \mathbf{Q}_j\mathbf{b} \wedge \forall y((y \neq \mathbf{p} \wedge \mathbf{Ma}y) \rightarrow \mathbf{Mb}y) \wedge \forall y((y \neq \mathbf{p} \wedge \neg \mathbf{Ma}y) \rightarrow \neg \mathbf{Mb}y)$

3. Sistema deductivo de árboles

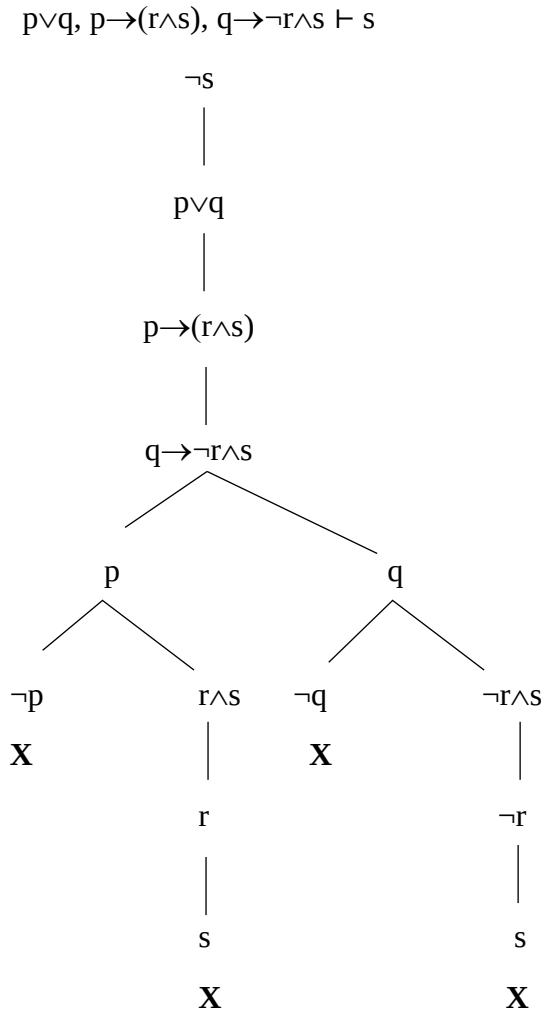
3.1 El método de árboles

3.1.1 Deducciones de árboles en lógica proposicional

El sistema deductivo de árboles es un conjunto de reglas a partir del cual, dados un conjunto de fórmulas (premisas) Γ y una formula *A* (conclusión) podemos construir un árbol de fórmulas. Según lo que ocurra en ese árbol diremos que *A* se deduce de Γ (en cuyo caso el árbol es una deducción de *A* a partir de Γ) o no. Por tanto, un árbol comprueba si un conjunto dado de fórmulas es satisfacible o no; además, se puede utilizar para verificar la validez de las fórmulas.

El objetivo principal de los árboles en lógica proposicional es intentar "dividir" fórmulas complejas en otras más pequeñas (atómicas) hasta que se produzcan pares complementarios de literales o que no sea posible una mayor expansión.

3.1.1.1 Ejemplo de árbol de lógica proposicional



3.1.2 Deducciones de árboles en lógica de primer orden

El planteamiento y elaboración de los árboles en lógica de primer orden es el mismo que el de la lógica proposicional. Lo único es que hay que extenderlo por la mayor capacidad expresiva que tiene la lógica de primer orden en comparación con la proposicional.

Para poder extender este método a la lógica de primer orden es necesario añadir las reglas correspondientes a los nuevos tipos de fórmulas que tenemos, es decir, las formulas con cuantificadores y con identidades[11]. Por tanto, lo que necesitamos ahora son cuatro reglas nuevas que nos indiquen que hay que hacer en una rama del árbol cuando aparecen fórmulas de cada uno de los cuatro nuevos tipos con los que nos podemos encontrar, esto es, de los tipos $\forall xA$, $\neg\forall xA$, $\exists xA$, $\neg\exists xA$.

3.1.2.1 Explicación de las reglas para las fórmulas cuantificadas

Cuantificador Universal

En el caso del cuantificador universal, $\forall xA$, la aplicación de su regla consiste en que ante una formula del tipo $\forall xA$ debemos añadir al final de cada una de las ramas en las que está esa fórmula todas las formulas del tipo $A[t/x]$, donde t es un término que estaba ya en la rama y, a continuación, repetir la fórmula $\forall xA$, con vistas a añadir más adelante las fórmulas $A[t/x]$ correspondientes a los nuevos términos que puedan ir apareciendo en la rama (hay que precisar que si en la rama no aparece ningún termino libre, debemos añadir al menos una formula del tipo $A[t/x]$, en cuyo caso da igual cual sea t). Por tanto, cuando nos encontramos con una fórmula de la forma $\forall xA$ añadimos a la rama todas las formulas del tipo $A[t/x]$ para todos los términos t que estaban previamente en la rama y cada vez que aparezca una constante nueva en la rama añadimos la formula correspondiente. Esto significa que, a diferencia de lo que ocurría hasta ahora, que una vez que se había aplicado a una formula del árbol la regla correspondiente nos podíamos olvidar de esa fórmula, de las formulas universales no podemos olvidarnos nunca: una vez que aparecen hay que tenerlas presentes una y otra vez hasta que se terminen todas las ramas en que aparecen.

Mientras que, en el caso de la negación del cuantificador universal para que una fórmula del tipo $\neg\forall xA$ sea satisfecha es necesario que algún objeto del dominio no

[11] En el presente trabajo nos limitaremos a presentar y utilizar un lenguaje de primer orden sin identidad.

satisfaga *A*, o sea que algún objeto del dominio satisfaga ¬*A*. Eso corresponde a la satisfacción de alguna fórmula del tipo ¬*A*[*t*/*x*], y por tanto es una fórmula de este tipo la que debe aparecer en la rama. La cuestión ahora es que, para que se satisfaga ¬∀*xA* debe haber algún objeto que satisfaga ¬*A*[*t*/*x*], pero no sabemos cuál y, en particular, no podemos afirmar que sea ningún objeto conocido, puesto que podría ser cualquiera. "Conocido" en términos de una deducción, significa simplemente que aparece en alguna fórmula de la deducción, así que la cuestión es que no podemos añadir una formula del tipo ¬*A*[*t*/*x*] donde *t* sea un término que ya aparece en la rama o en alguna premisa (incluso si esta premisa todavía no está en el árbol): **debe ser un término nuevo**.

Cuantificador Existencial

Por lo que se refiere al cuantificador existencial, necesitaremos también dos reglas que, como es de esperar, serán similares a las del universal, teniendo en cuenta la interdefinición entre ambos cuantificadores. Así, una formula del tipo ∃*xA* equivale a ∀*xA*, ¬∀*x*¬*A*, mientras que ¬∃*xA* equivale a ∀*x*¬*A*. Además, en el caso de la negación del cuantificador existencial debemos tener la misma precisión que en el caso de las formulas del tipo ∀*xA*: si en la rama no aparece ningún termino, debemos añadir al menos una fórmula del tipo ¬*A*[*t*/*x*], da igual cual sea el termino elegido).

Cuantificador Existencial

$\exists xA$

$A[t/x]$ (3)

(3) *t* es un término que no aparece libre ni en las premisas ni en la rama

Negación del Cuantificador Existencial

$\neg\exists xA$

$\neg A[t/x]$ (1)

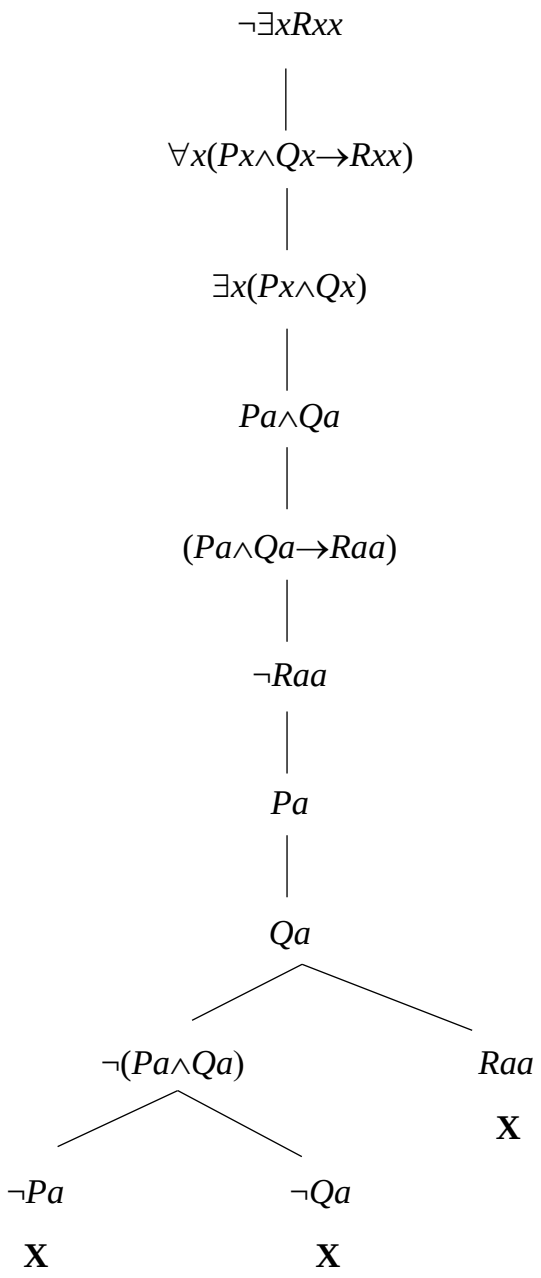
$\neg\forall xA$ (2)

(1) Para todos los términos *t* que aparecen libres en la rama

(2) la repetición es superflua; sólo tiene un valor mnemotécnico.

3.1.2.2 Ejemplo de árbol de lógica de primer orden

$$\forall x(Px \wedge Qx \rightarrow Rxx),\ \exists x(Px \wedge Qx) \vdash \exists x Rxx$$



3.2 Árboles infinitos en lógica de primer orden y su problemática

$$\forall x \exists y Pxy \vdash \neg \exists y Pay$$

(1)	$\neg\neg\exists y Pay$	
(2)	$\forall x \exists y Pxy$	
(3)	$\exists y Pay$	DN 1
(4)	Pab	V∃ 3 (y: b)
(5)	$\exists y Pay$	V∀ 2 (x: a)
(6)	$\exists y Pby$	V∀ 2 (x: b)
(7)	Pac	V∃ 5 (y: c)
(8)	$\exists y Pcy$	V∀ 2 (x: c)
(9)	Pbd	V∃ 6 (y: d)
(10)	$\exists y Pdy$	V∀ 2 (x: d)

ÁRBOL INFINITO

Si observamos detenidamente este árbol inconcluso nos damos cuenta en seguida de que la aplicación de las reglas no acabaría nunca en él. Los pasos (1) y (2) son, respectivamente, la negación de la conclusión y la premisa. En el paso (3) se aplica la regla de eliminación de la doble negación. (4) sale de (3) sustituyendo *y* por una constante nueva *b*. La aparición de *b* nos obliga a aplicar la regla a la formula (2), lo que da lugar a (6). (5) también es la aplicación de la regla a la fórmula (2), porque en el árbol tenemos también la constante *a*. Por tanto, (5) a su vez, exige otra constante nueva (7), que a su vez nos lleva a una nueva instanciación de la fórmula original (8), lo que nos llevara a otra constante nueva y así indefinidamente. Lo mismo ocurre con (6) dando lugar a (9) y (10). De esta manera, la aplicación de reglas puede ser infinita incluso aunque partamos de una única fórmula.

$\forall x \exists y Pxy \vdash \exists y \forall x Pxy$

(1)	$\neg \exists y \forall x Pxy$	
(2)	$\forall x \exists y Pxy$	
(3)	$\neg \forall x Pxa$	F∃ 1 (*y*: *a*)
(4)	$\exists y Pay$	V∀ 2 (*x*: *a*)
(5)	$\neg Pba$	F∀ 3 (*x*: *b*)
(6)	Pac	V∃ 4 (*y*: *c*)
(7)	$\neg \forall x Pxb$	F∃ 1 (*y*: *b*)
(8)	$\neg \forall x Pxc$	F∃ 1 (*y*: *c*)
(9)	$\exists y Pby$	V∀ 2 (*x*: *b*)
(10)	$\exists y Pcy$	V∀ 2 (*x*: *c*)
(11)	$\neg Pdb$	F∀ 7 (*x*: *d*)
(12)	$\neg Pec$	F∀ 8 (*x*: *e*)
(13)	Pbf	V∃ 9 (*y*: *f*)
(14)	Pcg	V∃ 10 (*y*: *g*)

ÁRBOL INFINITO

(1) y (2) son la negación de la conclusión y la premisa. (3) procede de (1) reemplazando *y* por *a*, una constante cualquiera, puesto que no hay ninguna en la rama. (4) procede de (2), usando la constante que ya tenemos. (5) procede de (3), y (6) de (4) usando constantes nuevas, como corresponde a la regla de la negación del cuantificador universal y del cuantificador existencial. Pero la aparición de esas dos constantes nuevas nos obliga a volver a usar las formulas (1) y (2), cada una de ellas con las dos constantes nuevas, lo que da lugar a (7), (8), (9) y (10), que, a su vez, dan lugar respectivamente a las formulas (11), (12), (13) y (14), en cada una de las cuales aparece una nueva constante como consecuencia de la aplicación de la regla correspondiente. Obviamente, esas constantes deben usarse en una nueva aplicación de las reglas de la negación del cuantificador existencial y del cuantificador universal correspondientes a las formulas (1) y (2), y así indefinidamente. Está claro que el árbol no se acabará nunca, pues las reglas nos obligarán a introducir más y más constantes nuevas.

Lo mismo ocurriría si nos encontrásemos con una única fórmula del tipo $\forall x \exists y Qxy$.

(1) $\forall x \exists y Qxy$

(2)	$\exists y Qay$	V∀ 1 (x: a)
(3)	Qab	V∃ 2 (y: b)
(4)	$\exists y Qby$	V∀ 1 (x: b)
(5)	Qbc	V∃ 4 (y: c)
(6)	$\exists y Qcy$	V∀ 1 (x: c)
(7)	Qcd	V∃ 6 (y: d)

ÁRBOL INFINITO

Como vemos aun partiendo de una sola fórmula tendríamos una rama infinita. Por tanto, podemos concluir que son las reglas de los cuantificadores la "causa" de esto, ya que en el caso de que haya un cuantificador existencial en una fórmula por regla hay que introducir siempre una constante nueva, mientras que en el caso del cuantificador universal sabemos que toda constante nueva que aparezca en el árbol tiene que volver a utilizarse en la fórmula sobre la que cuantifica.

Esto constituye una diferencia importante con lo que ocurría en lógica proposicional, donde, salvo en caso de que partiéramos de un conjunto infinito de premisas, los arboles siempre se acababan, tanto si se cerraban todas sus ramas como si no. Ahora ya no es así: un árbol en el que no todas las ramas se cierren puede ser infinito, incluso si partimos de un conjunto finito de premisas. Esto tiene que ver, como ya se ha dicho y demostrado en los dos apartados anteriores, con el hecho de que la lógica de primer orden es *indecidible*. Se dice que una lógica es *decidible* cuando existe un método que sirve siempre para establecer, dada cualquier fórmula (de esa lógica) si es válida o no. En la lógica proposicional, el método de árboles es un método de ese tipo (esto es, un método de decisión): basta construir un árbol a partir de la negación de la fórmula en cuestión; si se cierran todas las ramas, la fórmula era válida y, si no, no lo era. El método funciona porque el árbol, en lógica proposicional, siempre se termina, tanto si se cierran todas las ramas como si no, así que tenemos la seguridad de que en algún momento (al terminar de construir el árbol) sabremos la respuesta acerca de la fórmula que nos interesa.

Lo que acabamos de ver respecto a los árboles en lógica de primer orden indica que no sirven como método de decisión, pues en algunos casos no se acaban nunca, con lo que no siempre nos dan una respuesta. Esto no muestra que la lógica de primer orden no sea indecidible, sino simplemente que los árboles no son un método de decisión para ella.

Pero, ¿qué relación guarda esto con todo lo expuesto anteriormente?

3.3 ¿Por qué los árboles no son un método de decisión para la lógica de primer orden?

Sabemos que debido a la indecidibilidad no existe ningún procedimiento que nos diga si una fórmula de primer orden es válida o no. Claramente por medio del método de árboles podemos saber si esa fórmula es válida: realizamos el árbol, negando esa fórmula y si todas las ramas del árbol se cierran podemos concluir que dicha fórmula es válida. Si una de las ramas se quedara abierta y, por tanto, no se cerraría el árbol significa que la fórmula no es válida. Esto implicaría que si todos los árboles se tuviesen que cerrar entonces sí que tendríamos, efectivamente, un método de decisión.

Esto último, como hemos dicho anteriormente, queda bien reflejado en el caso de la lógica proposicional. Al realizar un árbol vamos aplicando una serie de reglas que nos indican como prolongar las ramas del árbol. Una rama está cerrada cuando contiene a la vez una formula y su negación. Mientras eso no ocurre, la rama está abierta. Un árbol está terminado cuando todas las ramas están cerradas, o se han aplicado ya todas las reglas posibles a todas las fórmulas del árbol. Por tanto, *A* es deducible de Γ cuando al hacer el árbol correspondiente todas las ramas se cierran. No lo es si el árbol correspondiente, una vez terminado, tiene alguna rama abierta.

Pero, la cuestión es que sabemos, como hemos visto en el caso de la lógica de primer orden, que hay árboles que pueden no cerrarse, simplemente porque no se acaban nunca, como es el caso de los árboles infinitos. ¿Qué ocurre en el caso de que no se cierren las ramas de un árbol?

1. Se han aplicado ya todas las reglas posibles a todas las fórmulas del árbol. Por lo cual, sabemos que *A* no es deducible de Γ.
2. Se han aplicado todas las reglas posibles a las fórmulas y en caso de no cerrarse todas las ramas, tendremos un árbol infinito en el que ya se han introducido todas las premisas en todas las ramas y se han aplicado todas las reglas posibles a todas las fórmulas, con lo que tendríamos un árbol terminado pero infinito. Sin

embargo, nunca sabríamos si la fórmula es válida o no, porque solo la rama infinita nos da una respuesta, y la rama completa no la tenemos en ningún momento finito.

Este segundo punto es el que nos interesa, porque si existiese algún método o alguna regla que nos permitiera reconocer a partir de un trozo finito de árbol, que aún no está acabado, que tal árbol va a ser infinito, entonces el problema de la decisión estaría resuelto. Pero simplemente con un trozo de ese árbol, o de una rama, no podemos saber de manera sistemática que vaya a ser infinito, porque no puede existir ninguna instrucción que nos permita detectarlo. Curiosamente, de forma intuitiva nosotros en muchos casos podemos saberlo ya que al realizar el árbol y aplicando las respectivas reglas, nos damos cuenta de que nos estamos metiendo en algún tipo de bucle o de proceso repetitivo que no va a terminar, como ocurre en los ejemplos que hemos visto. El problema es que no disponemos de una "herramienta" que reconozca, por medio de reglas, que esa rama, o árbol, va a ser infinita, de forma que podamos afirmar que teniendo estas condiciones, la rama se quedaría abierta, lo que nos permitiría dar por terminado el árbol, y tener así una respuesta negativa a la validez de la fórmula.
Esto es lo que, al final, afirma el teorema de Church: no hay ninguna manera que nos permita saber que una rama del árbol vaya a ser infinita a partir de un trozo finito del mismo.

¿Cómo podemos detectar que ese árbol será infinito? Hemos visto en el punto 3.2 que básicamente para que una rama sea infinita debe haber una fórmula que esté formada primero por un cuantificador universal y luego por un cuantificador existencial. Volvemos a proponer uno de los ejemplos. Si nos fijamos en esta fórmula $\forall x \exists y Qxy$, vemos como ésta es un claro ejemplo de todo lo que hemos expuesto hasta ahora. Lo primero que tendríamos que hacer es aplicar la regla del cuantificador universal para sustituir x por un término libre y luego aplicar la regla del existencial que nos daría una constante nueva. Teniendo dicha constante tendríamos que volver a usar la regla del universal para sustituir todos los términos sobre los que cuantifica y así recursivamente. Esto se debe únicamente a las reglas de los cuantificadores porque, como sabemos, cuando nos encontramos con una fórmula del tipo $\forall xA$ tendremos que añadir a la rama todas las formulas del tipo $A[t/x]$ para todos los términos t que estaban previamente en la rama y cada vez que aparezca una constante nueva en la rama añadimos la formula correspondiente.

Cada vez que apliquemos la regla del cuantificador existencial tendremos una constante nueva y por la regla del cuantificador universal tendremos que usarla para sustituir los otros términos. Pero, para que esto no ocurra, no podemos cambiar, sin más, unas reglas ya establecidas. Simplemente es así.

3.4 Comparación de los árboles infinitos con el *halting problem*

Con este "problema" de las reglas de los cuantificadores que generan árboles infinitos se podría establecer una relación con el problema de la parada. Recordemos que éste planteaba que si dada una MdT *M* y un *input n* la maquina se pararía o no. Podemos ver como esto podría aplicarse al caso de los arboles infinitos, o sea, saber si el árbol se cerraría o no.

En el caso de las máquinas de Turing si una máquina no se para es porque hay, al menos, una instrucción que se aplica de manera infinita. Es decir, si tuviéramos una cantidad finita de instrucciones la máquina las iría aplicando hasta acotarlas. Pero, si la máquina no se para tiene que haber una instrucción que se aplica un número infinito de veces. Mientras que, en el caso de los árboles, si un árbol no se cierra es porque hay alguna regla, o fórmula, que tiene que aplicarse un número infinito de veces, como es el caso de la regla del cuantificador universal.

Además, una máquina de Turing puede no pararse por una serie de motivos, como hemos visto en el punto 1.3. Uno de ellos es porque la máquina entra en bucle, es decir, que haría lo mismo uno y otra vez. Esto implica que hay una configuración de la máquina que aparece continuamente, o sea, si en una máquina una misma configuración aparece dos, o más, veces a lo largo de una ejecución, significa que el programa va a ser infinito: en el momento en que la máquina vuelve a estar en el mismo estado que al principio, volverá a hacer lo mismo otra vez y, claramente, aparecerá de nuevo la misma configuración. Lo cual implica que lo que ocurre en esa configuración se va a repetir indefinidamente.

Ocurre lo mismo en el caso de los árboles al tener los dos cuantificadores en un orden establecido, como se ha visto en los ejemplos propuestos. La diferencia con la MdT es que no tenemos la misma fórmula una y otra vez, pero son fórmulas totalmente idénticas: si sustituimos las constantes unas por otras se volverá a repetir la aplicación de las reglas infinitivamente.

De hecho podemos establecer con todos los ejemplos propuestos de los árboles infinitos una similitud con la configuración del problema de la parada. En el caso del *halting problema* se debe a la configuración de la máquina y en el caso de los árboles por el orden de los cuantificadores de las fórmulas propuestas. Lo que nos importa aquí es destacar la analogía que se da entre árboles con ramas infinitas y máquinas de Turing que no se paran: en ambos casos podemos ver ejemplos clarísimos de procesos que no van a terminar, lo que nos lleva a tener una visión intuitivamente clara del tipo de proceso que se da en una rama infinita de un árbol y en un proceso computacional que no se acaba, la existencia de algún tipo de instrucción que vamos a aplicar una y otra vez en condiciones similares. Y, sin embargo, las demostraciones de indecidibilidad nos enseñan que es imposible transformar esa intuición en una instrucción sistemática, en una regla fija.

Conclusión

Nos gustaría disponer de un procedimiento que nos permitiera determinar si una fórmula cualquiera de lógica de primer orden es válida o no, es decir, que el conjunto de esas fórmulas fuera decidible. El problema de determinar si existe un procedimiento que nos dé una respuesta concreta, sí o no, se la conoce como problema de la decisión.

La lógica de primer orden sería decidible syss fuera posible proporcionar un método que nos permitiera establecer la validez o no validez de una fórmula cualquiera. Ahora bien, el procedimiento proporcionado por tal método de decisión para la lógica de primer orden debería poder establecerse de manera mecánica sin ningún tipo de elemento intuitivo.

Podríamos decir que una de las claves para abordar el problema reside en cómo debemos entender el término "método" y como éste nos ayude a establecer la validez de una fórmula.

Esa cuestión es la que nos ha llevado a la presentación de las máquinas de Turing, como forma de análisis del concepto de método mecánico, o de resolución de problemas por medio de algoritmos y a su vez, a la tesis de Church-Turing, como justificación de la generalidad y suficiencia de dicho análisis para dar cuenta de la idea intuitiva de procedimiento de decisión. Hemos abordado además el problema de la parada, presentando no una demostración formal completa, sino una descripción lo más clara posible de la estructura básica de dicha demostración que permita comprender la idea básica que subyace a ella. Y algo similar hemos hecho a propósito de la demostración de indecidibilidad de la lógica de primer orden, presentando también en este caso la idea general y la estructura básica del razonamiento que demuestra que una solución al problema de la validez de las fórmulas de primer orden conllevaría una solución al problema de la parada. La consideración del método de árboles, que en lógica proposicional constituye en sí mismo un método decisión, en lógica de primer orden no lo es ni, a la luz del resultado de indecidibilidad, podría serlo, nos lleva a una reflexión acerca del concepto de indecidibilidad. Efectivamente, hemos tratado de ver la similaridad entre procesos computacionales que no terminan y árboles con ramas infinitas, y hemos visto que ambos casos nos suscitan la misma perplejidad: por una parte vemos que parece que entendemos claramente por qué ocurre eso y por otra los resultados formales nos dicen que no podemos describir con absoluta precisión ese por qué. Parece

que, de alguna forma, entendemos qué clase de cosa puede dar lugar a un proceso infinito (computación o construcción del árbol) o a una entidad infinita (la computación infinita completa o el árbol infinito completo, considerados como entidades infinitas en acto) pero no podemos sin embargo describir exactamente el tipo de procesos que generan eso porque, si pudiéramos, podríamos reconocerlos cuando se presentan (en una computación o en un árbol) y por tanto, resolver los problemas de decisión. Quizá podríamos concluir diciendo que lo que se pone de relieve es el problema de dilucidar nuestra comprensión de la infinitud.

Bibliografía

- Boolos, G., Burgess, J.P., Jeffrey, R. (2007). *Computability and Logic*. Cambridge: Cambridge University Press.
- Church, A. (1936a). An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 58(2): 345–363.
- Church, A. (1936b). A Note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1(1): 40–41.
- Church, A. (1937). Review of: On Computable Numbers with An Application to the Entscheidungsproblem by A.M. Turing. *Journal of Symbolic Logic*, 2(1): 42–43.
- Hilbert, D. (1902). Mathematical problems: Lecture delivered before de International Congress of Mathematicians at Paris in 1900. *Bulletin of the New York Mathematical Society*, 8(10): 437-479.
- Kleene, S.C. (1952). *Introduction to metamathematics*. Amsterdam: North-Holland Publishing Co.
- Mendelson, E. (1990). Second Thoughts about Church's Thesis and Mathematical Proofs. *The Journal of Philosophy*, 87 (5): 225-233.
- Smullyan, R.M. (1971). *First-Order Logic.* New York: Springer.
- Turing, A. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42 (1): 230–265.
- Turing, A. (1948). Intelligent Machinery, en *Collected Works* (Volume 1), Amsterdam: North-Holland, 1992.