Abstract

Este trabajo describe teórica y prácticamente los *quines*, definidos como programas sin input cuyo output es su propio código. Se demuestran los teoremas S_n^m y de recursión de Kleene, que garantizan la existencia de *quines* en todo lenguaje Turing completo. Luego se dan ejemplos concretos de *quines* en lenguajes de programación conocidos.

1 Introducción

Informalmente, un *quine* es un programa sin input cuyo output es su propio código. Se trata de un caso particular de una clase más general de fenómenos caracterizados por la *auto-referencialidad indirecta*. Decimos que existe auto-referencialidad indirecta cuando una sentencia se denota a sí misma de manera implícita a través de un código u otra forma adecuada de representación. En lenguaje natural, la diferencia entre auto-referencialidad directa e indirecta puede ilustrarse con las denotaciones "*yo*" y "*la persona cuyo DNI es* x", donde x es mi número de DNI. En sistemas formales, la auto-referencialidad directa es volátil en el sentido de que induce recursiones infinitas. (Piénsese, por ejemplo, en la definición de GNU.) La auto-referencialidad indirecta, por otra parte, puede eludir este problema: una sentencia lógica puede enunciar algo acerca de su propio número de Gödel, por ejemplo, sin inducir una recursión infinita.

Los *quines* tienen algunas aplicaciones prácticas, por ejemplo en producción de virus, pero en rigor su valor difícilmente exceda el de una curiosidad teórica. A pesar de esto, entendemos que tienen al menos un valor ilustrativo: ejemplifican de manera concreta una aplicación del teorema de la recursión de Kleene. Dicho teorema no es insignificante: como veremos más adelante, guarda una relación directa con la currificación funcional y con la posibilidad de transformar programas sintácticamente sin alterar su semántica.

2 Auto-referencialidad directa e indirecta

La auto-referencialidad, en todas sus formas, es un problema clásico en la lógica. La auto-referencialidad *directa*, en la que una sentencia habla explícitamente de sí misma, cuenta con mayor antigüedad y ha recibido un tratamiento más extensivo. Sus ejemplos más paradigmáticos son la paradoja

del mentiroso, la paradoja de Epiménides, y la menos conocida paradoja de Curry. La última, por ejemplo, permite demostrar cualquier sentencia φ usando reglas deductivas válidas. Considere por ejemplo la sentencia

$$\varphi := \text{El profesor Pagano nos pondrá un diez.}$$

y la siguiente sentencia asociada:

$\psi :=$ Si esta sentencia es verdad, el profesor Pagano nos pondrá diez.

Veamos que la existencia de esta sentencia auto-referencial nos permite demostrar φ. En primer lugar, notemos que si asumimos el antecedente "esta sentencia es verdadera", se sigue inmediatamente el consecuente φ. Por lo tanto, ψ es verdadera. Pero como ψ es verdadera, y ψ dice que de ser verdadera se sigue φ, tenemos que φ es verdadera. $\therefore$ El profesor Pagano nos pondrá diez.

Toda sentencia, verdadera o falsa, puede demostrarse de este modo, con lo cual el hecho de que φ sea realmente verdadera es incidental[1]. La solución de la paradoja de Curry no es clara: la conclusión es lógicamente impecable y se deriva estrictamente de la auto-referencialidad directa de ψ. Pero, habiendo ejemplificado los problemas que acarrea la auto-referencialidad directa, ¿qué hay de la auto-referencialidad indirecta? ¿Qué rol juega en la lógica en general y, más concretamente, la computación teórica?

El ejemplo más paradigmático de auto-referencialidad indirecta, que a la postre muestra la importancia crucial de este concepto en la computación teórica, es el teorema de incompletitud de Gödel. Si T es la aritmética de Peano (PA) o una extensión consistente de ella, entonces el teorema de incompletitud garantiza la existencia de una sentencia ψ tal que

$$\text{PA} \vdash \psi \leftrightarrow \neg \mathcal{P}_T\Big(\langle \psi \rangle \Big)$$

donde $\langle \psi \rangle$ es el número de Gödel de la sentencia ψ y $\mathcal{P}_T(x)$ es el predicado "x es deducible de la teoría T". En otras palabras, ψ es una sentencia que dice "no soy demostrable en T". La auto-referencialidad es indirecta en la medida en que ψ es equivalente a una sentencia que contiene $\langle \psi \rangle$.

La auto-referencialidad indirecta parece evadir las paradojas asociadas a la auto-referencialidad directa. Constituye, por lo tanto, una herramienta

[1] Último chiste del artículo.

lógica poderosa. Para comprender por qué un *quine* es una forma de autoreferencialidad indirecta, debemos presentarlo teóricamente como un caso particular del teorema de la recursión de Kleene.

3 Kleene

Sea $\mathcal{F}$ el conjunto de funciones parciales computables. La aritmetización de la sintaxis dada por Gödel establece una relación biyectiva entre el conjunto de programas (en el sentido de máquinas de Turing u otro modelo equivalente) y los números naturales. Por lo tanto, como $\mathcal{F}$ es recursivamente enumerable, podemos dar la enumeración

$$\varphi_1, \varphi_2, \dots$$

tal que φ_k es la función computada por el programa k.

3.1 Currificación y el teorema S_n^m

Un resultado significativo dado por Kleene es que la currificación de una función computable es computable. Informalmente, esto significa que existe un programa tal que, dado otro programa de n variables, devuelve un programa de $1 \leq m < n$ variables que es funcionalmente equivalente al programa original con el valor de las primeras (o últimas) $n-m$ variables fijas. Usualmente, este resultado es conocido como el teorema S_n^m.

Más formalmente, si $\varphi_p(u, x)$ es la función de dos argumentos computada por p, existe una función computable $S(k, p)$ tal que $\varphi_{S(k,p)}(x) = \varphi_p(k, x)$. Este resultado, que puede demostrarse dentro del paradigma de Turing o, tal vez más intuitivamente, utilizando un paradigma imperativo equivalente, se generaliza para funciones $\varphi(x_1, x_2, \dots, x_n)$ de n argumentos.

Theorem 1 (Teorema S_n^m) *Sea $\mathcal{P}$ un programa arbitrario. Existe una función primitiva recursiva $S_n^m : \mathbb{N}^m \times \mathbb{P}$ tal que*

$$\varphi_{\mathcal{P}}(x_1, \dots, x_n, y_1, \dots, y_m) \simeq \varphi_{\mathcal{S}(y_1,\dots,y_m,\mathcal{P})}(x_1, \dots, x_n))$$

Prueba. Interpretemos $\mathcal{P}$ como una concatenación de instrucciones en un paradigma imperativo equivalente al paradigma de Turing. Convengamos que las variables de un programa son un conjunto enumerable $v_1, v_2, \dots$, y que $\varphi_{\mathcal{P}}(x_1, \dots, x_n)$ se corresponde con ejecutar $\mathcal{P}$ desde el estado en que las variables $v_1, \dots, v_n$ toman los valores

$x_1, \ldots, x_n$. Sea $\mathcal{Q}_i : \mathbb{N} \to \mathbb{P}$ la función tal que $\mathcal{Q}_i(x)$ devuelve el programa que asigna a la variable i el valor x. Es fácil ver que $\mathcal{Q}_i$ es primitiva recursiva. Entonces, definimos

$$S_n^m(y_1, \ldots, y_n, \mathcal{P}) := \mathcal{Q}_{n+1}(y_1); \ldots; \mathcal{Q}_{n+m}(y_m); \mathcal{P}$$

donde $\alpha; \beta$ es la concatenación de las instrucciones α, β. La concatenación de palabras es primitiva recursiva y por lo tanto S_n^m es primitiva recursiva. Es evidente que $S_n^m(y_1, \ldots, y_n, \mathcal{P})$ es el programa que ejecuta $\mathcal{P}$ desde un estado en que las variables $n+1, \ldots, n+m$ toman los valores $y_1, \ldots, y_m$. Por lo tanto,

$$\varphi_{S_n^m(y_1,\ldots,y_m,\mathcal{P})}(x_1, \ldots, x_n) \simeq \varphi_{\mathcal{P}}(x_1, \ldots, x_n, y_1, \ldots, y_m) \blacksquare$$

El resultado teórico que más nos interesa por su relación con los *quines* es el teorema de la recursión de Kleene, cuya demostración depende del teorema S_n^m. El teorema de la recursión garantiza que, para toda $f \in \mathcal{F}$, existe un programa e tal que e y $f(e)$ computan la misma función, o equivalentemente $\varphi_e(x) \simeq \varphi_{f(e)}(x)$. El teorema se demuestra constructivamente.

Theorem 2 (Teorema de la recursión de Kleene) *Sea $f \in \mathcal{F}$. Entonces existe e tal que $\varphi_e(x) \simeq \varphi_{f(e)}(x)$.*

Prueba. Sea

$$\varphi_{f(S(u,u))}(x) =: g(u, x)$$

Como g es computable, existe un n tal que $g(u, x) = \varphi_n(u, x)$. Si definimos $e := S(n, n)$, resulta entonces

$$\begin{aligned}
\varphi_e(x) &= \varphi_{S(n,n)}(x) \\
&= \varphi_n(n, x) \\
&= g(n, x) \\
&= \varphi_{f(S(n,n))}(x) \\
&= \varphi_{f(e)}(x)
\end{aligned}$$

Por lo tanto, los programas e y $f(e)$ computan la misma función.

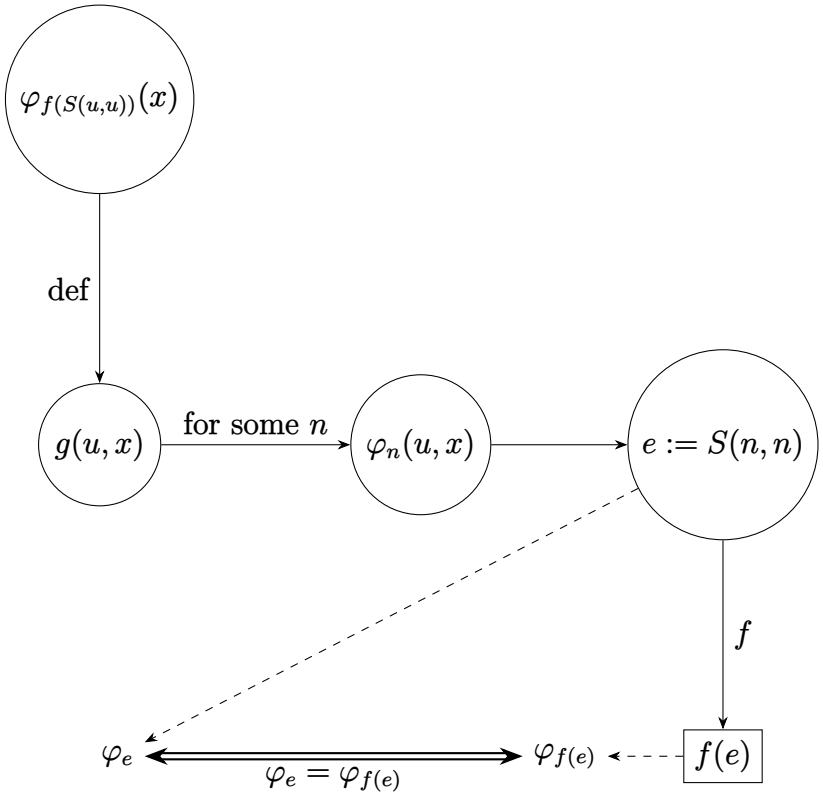

Figure 1: Diagrama ilustrando la construcción del punto fijo en el teorema de la recursión de Kleene

Intuitivamente, es importante tomar del teorema de la recursión de Kleene la idea de que podemos transformar cualquier programa en uno nuevo que, aunque difiera en su sintaxis, compute la misma función. Más aún, dicha transformación y su construcción son computables, y constituyen una forma de auto-referencialidad indirecta: el programa $f(e)$ tiene como input un programa funcionalmente equivalente (es decir, idéntico en términos de computabilidad), pero no necesariamente igual.

El teorema está intrínsecamente relacionado con el concepto de *quine* porque garantiza que un programa puede operar sobre una referencia de sí mismo. Si bien e y $f(e)$ computan la misma función, difieren en el hecho de que $f(e)$ tiene "conocimiento" de e. El teorema es lo suficientemente fuerte como para garantizar la existencia de *quines* en todo modelo de computación equivalente al de Turing.

Theorem 3 (Existencia de *quines*) *En todo lenguaje Turing completo, existe un quine.*

> **Prueba.** Puede darse un programa $\mathcal{Q}(x)$ cuyo output es x. Por el teorema de la recursión de Kleene, existe un programa e tal que $\mathcal{Q}(e) = e$. Por lo tanto, e es un programa cuyo output es e.
>
> $\therefore$ e es un *quine*.