3. Aplicações do processo diagonal

José Carlos Espírito Santo

No seu artigo de 1936 [T36], Turing deu uma resposta negativa ao *Entscheidungproblem*, o Problema da Decisão colocado por Hilbert e Ackermann, o qual perguntava, em termos modernos, se existia um algoritmo para decidir se uma dada fórmula da Lógica de 1ª Ordem é ou não é um teorema (ver [F19]). Este importante resultado para a Lógica assentava noutro resultado, a resposta negativa ao Problema da Paragem, não menos importante para disciplinas ainda não inventadas à altura: a teoria e a prática da programação de computadores. Com efeito, a resposta negativa ao Problema da Paragem, que neste artigo

chamaremos de Teorema de Turing, significa que não existe um algoritmo para decidir se, dado um programa, a execução desse programa desencadeada por um dado *input* termina ou não.

O Problema da Paragem é tratado na secção 8 do artigo de Turing de 1936, justamente intitulada "*Application of the diagonal process*". O processo diagonal, ou diagonalização, a que este título se refere é um método de demonstração inventado e aplicado pelo matemático Georg Cantor no último quartel do Séc. XIX, no contexto de investigações sobre a teoria de conjuntos, e a que Turing recorre para provar o seu resultado sobre o Problema da Paragem. O Teorema de Turing seguia-se a outros feitos notáveis (por exemplo, o Paradoxo de Russell, os Teoremas da Incompletude de Gödel), todos com o mesmo espírito de revelar paradoxos ou limitações dentro da Lógica, todos recorrendo à diagonalização na sua demonstração. Em contraste, Cantor havia aplicado a diagonalização para obter aquilo que hoje se designa por Teorema de Cantor, uma prova de que, na teoria de conjuntos, se pode passar sucessivamente de um conjunto infinito a outro ainda maior – um teorema, portanto, que não poderia ter um espírito mais expansivo.

Não surpreende que o "processo diagonal" tenha desde sempre despertado um fascínio que também guia este artigo. Uma coisa que se fez várias vezes (ver, por exemplo, [Y03]) foi identificar um teorema subjacente ao Teorema de Cantor e a todos os resultados limitativos da Lógica: a diagonalização seria utilizada uma única vez para demonstrar esse teorema, e cada um dos outros resultados sairia como consequência desse teorema comum, em virtude de razões adicionais específicas. Neste artigo, vamos identificar um teorema subjacente aos Teoremas de Cantor e Turing, para poder ver aquilo que Turing teve de acrescentar ao processo diagonal para obter o seu resultado – que foi sobretudo a ideia de máquina universal.

Por sua vez, a máquina universal depende da ideia de codificação: uma máquina pode ser reduzida a um número e codificada como *input* para outra máquina. Esse código é uma entidade que é duas coisas ao mesmo tempo: a máquina que ele codifica (e assim é "desmaterializada"), e o *input* de outra máquina. A importância da codificação no trabalho de Turing tem certamente inspiração no trabalho de Gödel [G31], com que Turing tomou contacto pouco antes de escrever o seu artigo: aí

vemos que um numeral, dentro de um sistema formal para a aritmética dos números naturais, pode representar um número que vai substituir uma variável numa fórmula, e pode simultaneamente codificar outra fórmula.

Imaginemos um tabuleiro de xadrez ou batalha naval onde, quer as linhas, quer as colunas desse tabuleiro estão numeradas. A diagonal principal é constituída pelas casas de coordenadas (1,1), (2,2), etc. O mesmo número identifica a linha e a coluna, e avançamos em duas dimensões em simultâneo, quando damos um passo nessa diagonal. No teorema subjacente que vamos propor, identificamos um conceito de codificação e sublinhamos o seu papel nas demonstrações através do processo diagonal. Veremos como a codificação permite a diagonalização, e como a diagonalização subentende uma codificação.

O que se segue, portanto, é a exposição de alguns resultados matemáticos. Mas espera-se que o leitor acompanhe o desenrolar dos acontecimentos. Tudo o que precisa de compreender à partida é pouco mais que os conceitos matemáticos de conjunto e função. Em três pontos, far-se-ão afirmações matemáticas cuja justificação não é possível nem oportuno dar em todo o detalhe: essas afirmações serão claramente isoladas. No resto do texto, o convite é para uma viagem ao jeito de Cantor e Turing e na sua peugada: munidos de matemática elementar, mas com a coragem e o engenho para desvendar os segredos do infinito e da computação.

Notação e terminologia matemáticas

Valores lógicos: $\mathfrak{v}$ (verdade) e $\mathfrak{f}$ (falsidade).

O conjunto $\{\mathfrak{v},\mathfrak{f}\}$ dos valores lógicos designa-se por $\mathfrak{L}$.

O conjunto $\{1,2,3,\ldots\}$ designa-se por $\mathbb{N}$ e os seus elementos dizem-se números naturais.

Sejam A um conjunto e a um objeto. A simbologia $a \in A$ significa que a é *elemento* ou *membro* de A. Seja ainda B outro conjunto. A simbologia $A \subseteq B$ significa que A é um *subconjunto* de B, e isto, por sua vez, quer dizer que cada elemento de A é também elemento de B. Por exemplo, o

conjunto dos números naturais pares é um subconjunto de $\mathbb{N}$, mas não vice-versa, pois cada número natural par é um número natural, mas há pelo menos um número natural que não é par. Por outro lado, tem-se sempre $A \subseteq A$.

Uma *função* do conjunto A no conjunto B é uma correspondência que associa, a cada elemento de A, um e um só elemento de B. As funções são habitualmente designadas pelas letras f, g, ou h. A simbologia $f:A \to B$ significa que f é uma função do conjunto A no conjunto B; neste caso, dizemos que A é o *domínio (de definição)* de f, que f *está definida* em A, e que f é de *tipo* $A \to B$. Se $f:A \to B$ e $a \in A$, então a diz-se um *argumento* de f; o único elemento de B que corresponde a a diz-se o *valor* de f em a (ou a *imagem* por f de a) e representa-se por $f(a)$; e diz-se que o valor $f(a)$ se obtém por *aplicação* de f a a. Por exemplo, se f for a função que associa, a cada número natural, o seu dobro, então $f:\mathbb{N} \to \mathbb{N}$, 3 é um argumento de f, $f(3) = 6$ e 6 é o valor de f em 3.

O conceito de função exclui, portanto, a situação em que um argumento tem duas imagens. Mas é perfeitamente possível que dois argumentos tenham a mesma imagem. Uma função onde esta situação não ocorre diz-se *injetiva*. Uma função injetiva associa imagens diferentes a argumentos diferentes. Por exemplo, a função $f:\mathbb{N} \to \mathbb{N}$ que associa, a cada número natural, o seu dobro é injetiva. Por outro lado, o conceito de função de A em B, digamos, não exige que cada elemento do conjunto B seja imagem de algum argumento da função. A uma função onde este requisito seja satisfeito vamos chamar *exaustiva* (em vez de sobrejetiva, que é o termo técnico usualmente empregue). Voltando ao mesmo exemplo, f não é exaustiva, porque as imagens por f são necessariamente números pares.

Se $f:\mathbb{N} \to B$, então f é uma *enumeração* de elementos de B. A terminologia justifica-se se imaginarmos a listagem $f(1), f(2), f(3), \ldots$ Nesta listagem $f(1)$ é o primeiro elemento, $f(2)$ o segundo, etc. Em geral, se $n \in \mathbb{N}$, então dizemos que $f(n)$ é o n-ésimo elemento, ou elemento de ordem n, da enumeração. Se a enumeração f for exaustiva, então cada $b \in B$ ocorre pelo menos uma vez na listagem $f(1), f(2), f(3), \ldots$; e se for injetiva, então cada $b \in B$ ocorre no máximo uma vez na mesma listagem, pelo que a listagem não tem repetições. Por exemplo, seja de novo

$f:\mathbb{N} \to \mathbb{N}$ a função tal que $f(n) = 2n$, para cada $n \in \mathbb{N}$. Esta função é uma enumeração não exaustiva e sem repetições de números naturais.

Uma *função parcial* do conjunto A no conjunto B é uma função de tipo $A' \to B$, onde A' é algum subconjunto de A, subconjunto esse que se diz o domínio da função parcial. A simbologia $f:A \hookrightarrow B$ significa que f é uma tal função (mas não especifica o domínio de f). Por exemplo, se f é a correspondência que associa, a cada número natural par, a sua metade, então f é uma função parcial de $\mathbb{N}$ em $\mathbb{N}$, ou, também vamos dizer, uma função de *tipo* $\mathbb{N} \hookrightarrow \mathbb{N}$. Naturalmente, na definição anterior de função parcial, é permitido o caso em que A' é afinal A; nesse caso, a função diz-se *total*. Isto significa que o conceito de função parcial engloba o conceito de função: uma função é uma função parcial cujo domínio de definição A' afinal compreende à totalidade de A; ou seja, uma função é uma função parcial que afinal é total.

Um *predicado* é uma função de tipo $A \to \mathfrak{L}$. Se o domínio do predicado for $\mathbb{N}$, o predicado diz-se *numérico*. Se nada for dito em contrário, os predicados neste artigo serão numéricos. Um exemplo de um predicado numérico é a função $\mathfrak{p}$ definida por: $\mathfrak{p}(n) = \mathfrak{v}$, se $n \leq 5$; e $\mathfrak{p}(n) = \mathfrak{f}$, caso contrário. Este é o predicado que representaríamos simplesmente pela fórmula $n \leq 5$.

Cantor

Em 1874 Cantor introduziu o seu método da diagonalização, para provar que os números transcendentes são incontáveis [C74]. Nas décadas seguintes desenvolveu a teoria dos conjuntos abstratos. Depois, aplicou o método da diagonalização para obter um resultado conhecido como teorema de Cantor [C91]. A nós, este teorema interessa na seguinte versão:

Teorema (Cantor): Não existe uma enumeração exaustiva do conjunto de todos os predicados numéricos.

Uma enumeração exaustiva de todos os predicados numéricos é, recorde-se, uma função cujo domínio é o conjunto $\mathbb{N}$, cujas imagens são predicados numéricos, de tal modo que cada predicado numérico é imagem de algum elemento do domínio. Em vez de predicados numéricos,

consideremos predicados com domínio num conjunto A arbitrário. Seja B o conjunto de todos esses predicados. De facto, o argumento de Cantor prova um teorema mais geral, que afirma não existir uma função $f:A \to B$ com a propriedade de exaustividade, isto é, tal que cada predicado em B seja imagem, pela função f, de algum elemento do domínio A. Agora repetimos o processo, com C o conjunto dos predicados com domínio B: não existe função $f:B \to C$ com a propriedade de exaustividade. E assim sucessivamente.

Qual o significado de tudo isto? A existência de uma função $f:X \to Y$ com a propriedade de exaustividade caracteriza precisamente a situação que, em termos informais, se descreveria como "o número de elementos de Y não excede o número de elementos de X". Mas a caracterização em termos da existência de uma certa função tem a vantagem de se aplicar a conjuntos arbitrários, possivelmente infinitos, e dispensar a definição de "número de elementos". Voltando aos conjuntos A e B do parágrafo anterior, a inexistência de uma função $f:A \to B$ com a propriedade de exaustividade tem a leitura de que o número de elementos de B *excede* o número de elementos de A. Ao iteramos o processo, concluímos que o número de elementos de C excede o número de elementos de B, e assim sucessivamente. Ora, na base desta cadeia está um conjunto A arbitrário. Se A for um conjunto infinito, obtemos que, apesar da infinitude de A, o número de elementos de B excede o número de elementos de A. Certamente, B é um conjunto infinito, mas, no sentido preciso dado pelo teorema de Cantor, "B é mais infinito do que A". E, a seguir, vem o conjunto C, que é "mais infinito do que B", e assim sucessivamente.

A demonstração do Teorema de Cantor é curta mas engenhosa. E estas observações que acabámos de fazer são matematicamente simples. Espanta que estejam ao alcance da razão humana, a uma distância de aparência tão curta, consequências tão dramáticas para a teoria matemática do infinito.

Turing

Para dar uma solução negativa ao Problema da Decisão, Turing precisava de sustentar a inexistência de um algoritmo para determinada tarefa. Milénios de História haviam produzido inúmeros *exemplos* de

algoritmos. Mas a tarefa de Turing tinha uma exigência inédita, nada menos que dar uma definição de algoritmo e assim delimitar matematicamente a totalidade dos algoritmos e determinar os limites da computabilidade, para poder argumentar que o Problema da Decisão estava para lá desses limites. O ainda estudante Turing enfrentou corajosamente o desafio e deu a sua definição em termos de um conceito de máquina – aquilo que hoje se designa por máquina de Turing.

O conceito de máquina de Turing determina o conceito de *função computável* de tipo $\mathbb{N} \hookrightarrow \mathfrak{L}$. Como não vamos repetir aqui os detalhes da definição de máquina de Turing, a definição de função computável, na medida em que depende do conceito de máquina de Turing, não vai ser dada em todo o detalhe. Intuitivamente, uma função computável é uma função para a qual existe um algoritmo (máquina de Turing) que calcula o valor da função para qualquer argumento. Sejamos um pouco mais precisos.

Uma máquina de Turing M determina (ou calcula) uma função, designada por f_M, de tipo $\mathbb{N} \hookrightarrow \mathfrak{L}$, do seguinte modo. Dado $n \in \mathbb{N}$, fornecemos a M uma representação convencionada de n como *input*. A máquina executa a partir desse *input* e uma de duas situações podem acontecer: (i) a execução termina, e nesse caso lê-se, de uma forma previamente convencionada, um *output* $\mathfrak{I} \in \mathfrak{L}$ a partir da configuração em que a máquina parou, e define-se $f_M(n) = \mathfrak{I}$; (ii) a execução não termina, e nesse caso deixa-se $f_M(n)$ indefinido.

Uma função computável é a função calculada por alguma máquina de Turing. Portanto, uma função f de tipo $\mathbb{N} \hookrightarrow \mathfrak{L}$ é computável se existir uma máquina de Turing M tal que $f = f_M$.

Por exemplo, o predicado "n é par?" é computável. Se a convenção for representar na máquina os números naturais em notação binária, consideremos M a máquina que procura o dígito menos significativo e produz um *output* consoante esse dígito é 0 ou 1. A função f_M é o desejado predicado – a função que, dado $n \in \mathbb{N}$, devolve $\mathfrak{p}$ ou $\mathfrak{f}$ conforme n é par ou não.

Um predicado numérico pode não ser apenas um predicado sobre números, se os números *codificarem* outras entidades – por exemplo as próprias máquinas de Turing.

Para a determinação de cada máquina de Turing, precisamos de especificar, grosso modo, o conjunto finito de instruções que a máquina executa, sendo que cada instrução da máquina de Turing é também finitamente especificável. Assim, a especificação de uma máquina de Turing M requer uma quantidade finita de informação, que pode ser codificada num número natural n_M – de modo que a máquinas diferentes correspondam códigos diferentes.

Será importante, no que se segue, formar, para cada máquina M, o par ordenado (n_M, f_M), e por fim colecionar todos estes pares num conjunto, que vamos designar por #. Se $(n, f) \in \#$, vamos habitualmente escrever $n = \#f$.

A codificação de máquinas de Turing pode ser feita de tal modo que o predicado "n é código de alguma máquina de Turing?" é computável. Um predicado muito mais difícil é: "n é código de uma máquina de Turing cuja execução a partir do *input* 1 termina?". Será computável este predicado? Uma resposta afirmativa implicaria exibir um algoritmo (máquina de Turing) que, a partir apenas do código $n = n_M$ de uma máquina M (o qual é apenas uma representação das instruções de M), respondesse ɒ ou ʃ a uma questão acerca de uma determinada execução dessa máquina.

Vejamos outro exemplo. Seja u a função de tipo $\mathbb{N} \hookrightarrow \mathfrak{Q}$ definida por:

- $u(n) = f(n)$, se $n = \#f$ *e* $f(n)$ *está definido*
- $u(n)$ *está definido, caso contrário*

A condição "$n = \#f$ *e* $f(n)$ *está definido*" é equivalente à condição "existe máquina de Turing M tal que $n = n_M$ e a execução de M a partir do *input* n termina". Um dos resultados mais importantes do artigo de 1936 é o seguinte:

Facto 1: u é computável.

Este facto decorre da existência da chamada máquina de Turing *universal*. Designemos esta função u por *função universal*.

Vejamos uma variação, aparentemente inocente, da função anterior. O *predicado da paragem* é o predicado $t:\mathbb{N} \hookrightarrow \mathfrak{Q}$ definido por

- $\mathfrak{t}(n) = \mathfrak{v}$, se $n = \#f$ *e* $f(n)$ *está definido*
- $\mathfrak{t}(n) = \mathfrak{f}$, *caso contrário*

Se este predicado fosse computável, teria de existir um algoritmo (máquina de Turing) para *decidir* (responder $\mathfrak{v}$ ou $\mathfrak{f}$) sobre a terminação da execução de M a partir do *input* $n = n_M$. Ora, o Teorema de Turing diz que tal algoritmo não existe.

Teorema (Turing): O predicado da paragem não é computável.

Teorema da diagonalização

Vamos agora identificar um teorema que subjaz aos Teoremas de Cantor e Turing e é demonstrado por diagonalização.

Seja $\mathfrak{D}$ o conjunto de todas as funções parciais de tipo $\mathbb{N} \hookrightarrow \mathfrak{L}$. Fixemos um subconjunto $\mathfrak{F}$ de $\mathfrak{D}$.

Uma *codificação* de $\mathfrak{F}$ é um conjunto – habitualmente designado por # – de pares ordenados (n, f), em que $n \in \mathbb{N}$ e $f \in \mathfrak{F}$, conjunto esse satisfazendo dois requisitos:

i) Toda a função $f \in \mathfrak{F}$ tem um código; ou seja, para toda a função $f \in \mathfrak{F}$, existe $n \in \mathbb{N}$ tal que $(n, f) \in \#$.
ii) Se n é código de f e f', então f e f' concordam no argumento n; ou seja, se $(n, f) \in \#$ e $(n, f') \in \#$, então f e f' estão ambas definidas em n e o valor de ambas as funções é igual nesse argumento, ou f e f' estão ambas indefinidas em n.

Em vez de $(n, f) \in \#$, vamos habitualmente escrever $n = \#f$ (ler: n é código de f).

Seja $\mathfrak{n}:\mathfrak{L} \hookrightarrow \mathfrak{L}$ a função que troca os valores lógicos, isto é, $\mathfrak{v}$ envia $\mathfrak{f}$ em e vice-versa. O nome $\mathfrak{n}$ é para sugerir "negação".

Seja $f \in \mathfrak{F}$. A partir desta função, pode fabricar-se uma outra função g, do mesmo tipo, a que vamos chamar a *negação* de f. Dado $n \in \mathbb{N}$, aplica-se f a este argumento. Se $f(n)$ não está definido, também $g(n)$ não está; caso contrário, $f(n)$ é um valor lógico, e o valor lógico $g(n)$ é definido como sendo $\mathfrak{n}(f(n))$.

Ora pode dar-se o seguinte caso notável: sempre que fabricamos g a partir de $f \in \mathfrak{F}$ segundo o modo descrito, o resultado é ainda uma função de $\mathfrak{F}$. Nesse caso, diz-se que $\mathfrak{F}$ é *fechado* para a operação de negação.

Seja g função parcial de tipo $\mathbb{N} \hookrightarrow \mathfrak{L}$. Dizemos que g *respeita* uma codificação # se g satisfaz as duas condições seguintes, para cada $f \in \mathfrak{F}$:

(A) $g(n) = \mathfrak{v}$, se $n = \#f$ e $f(n) = \mathfrak{v}$,
(B) $g(n) = \mathfrak{f}$, se $n = \#f$ e $f(n) = \mathfrak{f}$.

Estas duas condições podem ser combinadas numa só: se $n = \#f$ e $f(n)$ está definido, então $g(n) = f(n)$. Talvez assim fique mais clara a razão de g "respeitar" #: perante um argumento n, e caso n codifique a função f, o valor de g para esse argumento não é, por assim dizer, escolhido livremente, antes é o valor de f para esse argumento, quando este último valor está definido.

É simples definir funções satisfazendo (A) e (B). Um exemplo importante é o predicado $\mathfrak{a}$ definido por:

- $\mathfrak{a}(n) = \mathfrak{v}$, se $n = \#f$ e $f(n) = \mathfrak{v}$
- $\mathfrak{a}(n) = \mathfrak{f}$, caso contrário

Vamos designar este predicado por *predicado de auto-aceitação* (relativo à codificação #). A palavra "aceitação" advém de podermos dizer que *f aceita n* quando $f(n) = \mathfrak{v}$. Assim, satisfazem o predicado de auto-aceitação aqueles números n que codificam uma função f que aceita o próprio código $n = \#f$.

Isto é tudo quanto precisamos para obter o desejado teorema:

Teorema 1: Seja $\mathfrak{D}$ o conjunto de todas a funções parciais de tipo $\mathbb{N} \hookrightarrow \mathfrak{L}$. Seja $\mathfrak{F}$ um subconjunto de $\mathfrak{D}$ que admite uma codificação # e é fechado para a operação de negação. Então, nenhuma função parcial g de tipo $\mathbb{N} \hookrightarrow \mathfrak{L}$ satisfaz simultaneamente as três condições:

1. $g \in \mathfrak{F}$,
2. g *respeita* #,
3. g é total.

Por outras palavras: se uma função $g \in \mathfrak{F}$ *respeita* #, então g não é total (existe $n \in \mathbb{N}$ tal que $g(n)$ não é $\mathfrak{v}$ nem $\mathfrak{f}$).

Uma consequência imediata é a seguinte: um predicado numérico que respeite # não pertence a $\mathfrak{F}$. Por exemplo:

Corolário: O predicado de auto-aceitação não pertence a $\mathfrak{F}$.

Demonstração do Teorema: Seja g: $\mathbb{N} \hookrightarrow \mathfrak{L}$ em $\mathfrak{F}$ satisfazendo (A) e (B). Seja h a negação de g. Isto significa que:

i) $h(n) = \mathfrak{f}$, se $g(n) = \mathfrak{v}$,
ii) $h(n) = \mathfrak{v}$, se $g(n) = \mathfrak{f}$,
iii) $h(n)$ *está indefinido, se* $g(n)$ *está indefinido*.

Como $\mathfrak{F}$ é fechado para a operação de negação, $h \in \mathfrak{F}$, pelo que h tem direito a um código $k = \#h$. Este código é um número natural onde g não está definida. Vejamos porquê.

De i, ii, (A) e (B) segue, para cada $f \in \mathfrak{F}$:

(C) $h(n) = \mathfrak{f}$, se $n = \#f$ e $f(n) = \mathfrak{v}$,
(D) $h(n) = \mathfrak{v}$, se $n = \#f$ e $f(n) = \mathfrak{f}$.

Qual o valor de h para o argumento k? Consideremos os casos particulares de (C) e (D) resultantes de escolher f como sendo *a própria função* h. Por um lado, se $h(k) = \mathfrak{v}$, concluímos, pelo referido caso particular de (C), que $h(k) = \mathfrak{f}$; donde, $h(k)$ não é $\mathfrak{v}$. Por outro lado, se $h(k) = \mathfrak{f}$, concluímos, pelo referido caso particular de (D), que $h(k) = \mathfrak{v}$; donde $h(k)$ não é $\mathfrak{f}$. Portanto, $h(k)$ não é $\mathfrak{v}$ nem $\mathfrak{f}$. Mas, então, devido a i e ii, também $g(k)$ não é $\mathfrak{v}$ nem $\mathfrak{f}$. **QED**

Processo de diagonalização

A demonstração do Teorema 1 usa o "processo diagonal", a técnica de diagonalização inventada por Cantor. Porquê "diagonalização"? Imaginemos uma tabela infinita com uma linha e uma coluna por cada número natural.

	1	2	3	…	□	…	□	…	□	…
1										
2										
3										
…				…						
□					I		I		π(I)	
…						…				
□										
…								…		
□							?		?	
…										

Cada entrada desta tabela é identificada por um par de coordenadas (i, j), com i, (respetivamente j) o número natural que identifica a linha (respetivamente coluna) em que a entrada está. A *diagonal* da tabela (ver as entradas sombreadas) são as entradas com coordenadas (i, j) em que $i = j$.

Se $j = \#f$, a entrada (i, j) define-se assim: se $f(i)$ está definido, $(i, j) = f(i)$; caso contrário, pomos "?" na entrada (i, j). Portanto, se $j = \#f$, a j-ésima coluna representa a função f. Se j não é código de uma função, a j-ésima coluna é deixada em branco.

Seja $g: \mathbb{N} \hookrightarrow \mathfrak{L}$ em $\mathfrak{F}$ satisfazendo (A) e (B) e seja $m = \#g$. Decorre das condições (A) e (B), e de $m = \#g$, que: dado um número natural n, se a entrada (n, n) contiver um valor lógico, a entrada (n, m) contém o mesmo valor lógico (se a entrada (n, n) for "?", nada é exigido à entrada (n, m)). Então, para o argumento $n = k$, temos:

> (*) se a entrada (k, k) contiver um valor lógico, a entrada (k, m) contém o mesmo valor lógico.

Sejam h a negação de g e $k = \#h$. Agora, por momentos, k desempenha o papel de código de h. Dado um número natural n, decorre da definição de h que cada uma das entradas (n, m) e (n, k) contém um valor lógico

se e só se a outra contiver; e, nesse caso, o valor lógico contido em (n, k) resulta de aplicar a operação de negação ao valor lógico contido em (n, m). Então, para o argumento $n = k$ (k volta ao papel de argumento), segue facilmente:

> (**) se a entrada (k, k) contiver um valor lógico, a entrada (k, m) contém a negação desse valor lógico.

De (*) e (**) segue que (k, k) não contém um valor lógico.

Grosso modo, a m-ésima coluna contém uma cópia dos valores lógicos da diagonal, enquanto a k-ésima coluna contém a negação desses valores lógicos. Ora a diagonal e a k-ésima coluna intersetam-se na entrada (k, k) pelo que esta entrada não pode conter valor lógico (caso contrário, esse valor lógico teria de ser igual à sua negação).

Demonstração dos teoremas de Cantor e Turing

Como prometido, vamos ver agora que os Teoremas de Cantor e Turing são consequências do Teorema 1.

Demonstração do Teorema de Cantor: Seja $\mathfrak{F}$ o conjunto de todos os predicados. Suponhamos que havia uma enumeração exaustiva φ de $\mathfrak{F}$. Consideremos o gráfico de φ, o conjunto dos pares $(n, \varphi(n))$. Este conjunto, designemo-lo por #, definiria uma codificação de $\mathfrak{F}$: o requisito I seria satisfeito porque φ é exaustiva; o requisito II seria satisfeito porque φ é uma função (se $(n, f) \in \#$ e $(n, f') \in \#$, então $f' = f''$). Por outro lado, $\mathfrak{F}$ é fechado para a operação de negação, porque a negação de um predicado é um novo predicado. Do Teorema 1 seguiria que o predicado $\mathfrak{a}$ de auto-aceitação não pertence a $\mathfrak{F}$, o que é absurdo. A hipótese da existência de uma enumeração exaustiva de $\mathfrak{F}$ conduziu-nos a uma contradição. Logo, não existe tal enumeração. **QED**

Com vista a provar o Teorema de Turing, seja agora $\mathfrak{F}$ o conjunto de todas as funções computáveis de tipo $\mathbb{N} \hookrightarrow \mathfrak{L}$. Um resultado técnico simples é o seguinte:

Facto 2: $\mathfrak{F}$ é fechado para a operação de negação.

Recordemos o conjunto #, formado por todos os pares ordenados (n_M, f_M). Este conjunto é uma codificação de $\mathfrak{F}$: o requisito I é satisfeito porque toda a função computável é da forma f_M; o requisito II é satisfeito por que, se $n = \#f$ e $n = \#f'$, então $n = n_M$, para uma máquina M bem definida, e segue que $f = f_M = f'$.

Recordem-se a função universal u e o predicado da paragem $\mathfrak{t}$, referido no Teorema de Turing. Seja o predicado $\mathfrak{b}$ definido por:

- $\mathfrak{b}(n) = u(n)$, *se* $\mathfrak{t}(n) = \mathfrak{v}$
- $\mathfrak{b}(n) = \mathfrak{f}$, *nos restantes casos*

Note-se que $\mathfrak{b}$ é uma variante de u, onde o predicado $\mathfrak{t}$ é usado para antecipar a não-terminação.

Facto 3: Se $\mathfrak{t}$ for computável, $\mathfrak{b}$ também o é.

Um esboço da prova deste facto é o seguinte: o predicado $\mathfrak{b}$ resulta de uma espécie de "composição" das funções $\mathfrak{t}$ e u. Essa operação de "composição" preserva computabilidade: se aplicada a funções computáveis, produz uma função computável. Sabendo que u é computável, concluímos: se $\mathfrak{t}$ fosse computável, o resultado dessa "composição" também o seria.

Por fim, é fácil verificar que $\mathfrak{a}$ e $\mathfrak{b}$ são, afinal, a mesma função: usando sucessivamente as definições de $\mathfrak{b}$, u, $\mathfrak{t}$ e $\mathfrak{a}$, prova-se que $\mathfrak{b}(n) = \mathfrak{v}$ se, e só se, $\mathfrak{a}(n) = \mathfrak{v}$. Assim, vemos que, através da "composição" com o predicado da paragem, a função universal é transformada no predicado de auto-aceitação.

Demonstração do Teorema de Turing: Do Teorema 1 segue que o predicado de auto-aceitação $\mathfrak{a}$ não pertence a $\mathfrak{F}$. Então $\mathfrak{b}$ não é computável (porque $\mathfrak{a}$ e $\mathfrak{b}$ são a mesma função e $\mathfrak{F}$ é o conjunto das funções computáveis). Logo, $\mathfrak{t}$ também não é computável, devido ao Facto 3. **QED**

Referências bibliográficas

[C74] G. Cantor, *Ueber eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen*. Journal für die Reine und Angewandte Mathematik, 77 (77): 258-262, 1874

[C91] G. Cantor, *Ueber eine elementare Frage der Mannigfaltigkeitslehre*. Jahresbericht der Deutschen Mathematiker-Vereinigung. 1: 75-78, 1891

[G31] K. Gödel. Über *formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I*. Monatshefte für Mathematik und Physik, 38: 173-198, 1931. Tradução: *On Formally Undecidable Propositions of Principia Mathematica and Related Systems I*. Dover 1992.

[F19] F. Ferreira. *O problema da decisão e a máquina universal de Turing*. Neste volume.

[T36] A. Turing. *On computable numbers, with an application to the Entscheidungproblem*. Proceedings of the London Mathematical Society, 42(2):230-265, 1936.

[Y03] N. S. Yanofsky, *A universal approach to self-referencial paradoxes, incompleteness and fixed points*, Bulletin of Symbolic Logic, 9(3), 362-386, 2003.