UMA FORMA DIFERENTE DE FAZER CONTAS

LUÍS CRUZ-FILIPE

Resumo. Para muitos matemáticos, a Matemática é fundamentalmente um jogo. Neste artigo, apresentamos exactamente isso — apenas um jogo, com regras extremamente simples, que se descobre afinal ser um modelo teórico do funcionamento de um computador tão válido como qualquer outro.

1. Introdução

A Matemática é, por excelência, a ciência que faz as suas próprias regras. Como tal, muitas das áreas da Matemática nascem como verdadeiros jogos, em que os matemáticos são jogadores a explorar até onde é que conseguem ir com as regras que definiram.

Neste pequeno apontamento, vamos apresentar um jogo aparentemente muito simples: o Cálculo de Combinadores. Originalmente concebido por Moses Schönfinkel e Haskell Curry nos anos vinte, é um sistema extremamente simples — que tem todo o poder computacional do mais sofisticado computador existente nos nossos dias.

2. O jogo dos cartões

Antes de introduzirmos o jogo dos combinadores, vamos ver um outro jogo parecido. Para este jogo, a única coisa de que precisamos é de um conjunto de cartões empilhados e de uma caneta. Os cartões contêm instruções relativas aos cartões seguintes da pilha; um passo do jogo consiste em ler a instrução do primeiro cartão, executá-la e retirar o cartão da pilha. O processo repete-se até não ser possível executar a instrução do primeiro cartão da pilha. A pilha nunca pode ficar vazia.

Exemplos possíveis de instruções seriam: "retirar o cartão seguinte"; "trocar os próximos dois cartões"; "copiar esta instrução para um cartão novo e adicioná-lo a seguir ao segundo cartão da pilha"; "executar a instrução do terceiro cartão da pilha"; e assim sucessivamente. Um cartão pode também conter uma sequência de instruções; nesse caso, quando o cartão chega ao topo da pilha, é substituído por cartões separados contendo cada uma dessas instruções.

A última restrição é que as instruções têm de se referir a cartões concretos contados a partir do início da pilha; não é permitido, por exemplo, uma instrução como "duplicar o último cartão da pilha". (Daí estarmos a trabalhar com uma pilha: só podemos olhar para os elementos do topo.)

Vejamos um exemplo. Suponhamos que começamos com quatro cartões contendo as seguintes instruções.

(1) Trocar os dois cartões seguintes.
(2) Eliminar o segundo cartão a seguir a este.
(3) Duplicar o cartão a seguir a este.
(4) Copiar a(s) instrução(ões) do próximo cartão para um novo cartão e adicioná-lo em terceiro lugar na pilha.

Quando retiramos o primeiro cartão da pilha e seguimos a instrução lá escrita, a pilha passa a conter os seguintes cartões.

(1) Duplicar o cartão a seguir a este.
(2) Eliminar o segundo cartão a seguir a este.
(3) Copiar a(s) instrução(ões) do próximo cartão para um novo cartão e adicioná-lo em terceiro lugar na pilha.

Processando agora o cartão no topo, obtemos a pilha seguinte.

(1) Eliminar o segundo cartão a seguir a este.
(2) Eliminar o segundo cartão a seguir a este.
(3) Copiar a(s) instrução(ões) do próximo cartão para um novo cartão e adicioná-lo em terceiro lugar na pilha.

Finalmente, retirando o primeiro cartão e cumprindo as suas instruções, ficamos com a pilha reduzida a um cartão.

(1) Eliminar o segundo cartão a seguir a este.

Uma vez que não podemos executar esta instrução, o jogo termina.

Vejamos outro exemplo, agora com instruções múltiplas no mesmo cartão. A pilha inicial é a seguinte.

(1) Trocar os próximos dois cartões.
(2) Eliminar o próximo cartão da pilha. Trocar os próximos dois cartões.
(3) Copiar a(s) instrução(ões) do cartão seguinte para o cartão a seguir a esse.
(4) Duplicar o cartão a seguir ao próximo.

Ao retirar o primeiro cartão e seguir a sua instrução, passamos a ter a pilha seguinte.

(1) Copiar a(s) instrução(ões) do cartão seguinte para o cartão a seguir a esse.
(2) Eliminar o próximo cartão da pilha. Trocar os próximos dois cartões.
(3) Duplicar o cartão a seguir ao próximo.

O próximo cartão manda copiar as instruções do cartão seguinte para o cartão a seguir a esse. Uma vez que o próximo cartão tem duas instruções, a nova instrução do cartão 3 deve passar a ser a que lá está seguida daquela sequência. Vamos usar parênteses para indicar essa prioridade; a continuação do exemplo tornará clara a necessidade deste cuidado.

(1) Eliminar o próximo cartão da pilha. Trocar os próximos dois cartões.
(2) Duplicar o cartão a seguir ao próximo. (Eliminar o próximo cartão da pilha. Trocar os próximos dois cartões.)

O primeiro cartão agora contém duas instruções, pelo que o vamos substituir por dois cartões separados.

(1) Eliminar o próximo cartão da pilha.
(2) Trocar os próximos dois cartões.
(3) Duplicar o cartão a seguir ao próximo. (Eliminar o próximo cartão da pilha. Trocar os próximos dois cartões.)

O primeiro cartão manda-nos eliminar o cartão seguinte, pelo que a pilha fica reduzida a um cartão com uma sequência de instruções. Substituindo-o por dois cartões, obtemos a pilha seguinte.

(1) Duplicar o cartão a seguir ao próximo.
(2) Eliminar o próximo cartão da pilha. Trocar os próximos dois cartões.

Uma vez que não podemos executar a instrução do primeiro cartão, o jogo termina. Note-se que se não tivéssemos tido o cuidado de indicar que as instruções no segundo cartão eram um bloco, o resultado do jogo teria sido diferente.

O próximo exemplo deixa-se ao cuidado do leitor interessado e ilustra que este jogo tem mais potencial do que à primeira vista possa parecer...

(1) Duplicar a instrução do segundo cartão da pilha (no mesmo cartão).
(2) Não fazer nada.

(3) Duplicar a instrução do segundo cartão da pilha (no mesmo cartão). Não fazer nada.

3. O jogo dos combinadores

Para um matemático, o jogo que descrevemos na secção anterior é desnecessariamente sofisticado — usa objectos como cartões! Vamos então abstrair um pouco e ver como poderíamos jogar o mesmo jogo apenas manipulando letras numa folha de papel.

Para começar, vamos representar cada instrução escrita num cartão por uma letra. Por exemplo, vamos usar a letra **T** para representar a instrução "trocar os dois cartões seguintes". A sequência de cartões fica então reduzida a uma sequência de letras; quando um cartão tem várias instruções, usamos parênteses para indicar que se trata de um único cartão — sabendo que podemos remover parênteses no início da sequência.

Vamos ainda usar a notação $A \longrightarrow B$ para indicar que a sequência de cartões (letras) A se transforma na sequência B quando aplicamos um passo do jogo. Quando for adequado, usaremos ainda $A \longrightarrow^* B$ para indicar que A se transforma em B através duma sequência de passos.

Vejamos então o nosso primeiro exemplo de há pouco sob esta perspectiva, usando as letras seguintes.

- **T** para "trocar os dois cartões seguintes"
- **K** para "eliminar o segundo cartão a seguir a este"
- **M** para "duplicar o cartão a seguir a este"
- **P** para "copiar o próximo cartão para um novo cartão e adicioná-lo em terceiro lugar"

A sequência inicial é então **TKMP** e a sequência de jogo pode ser escrita como

$$\mathbf{TKMP} \longrightarrow \mathbf{MKP} \longrightarrow \mathbf{KKP} \longrightarrow \mathbf{K}\,.$$

Para o segundo exemplo, precisamos de mais letras. Vamos usar as seguintes.

- **f** para "eliminar o próximo cartão da pilha"
- **A** para "copiar a(s) instrução(ões) do próximo cartão para o cartão a seguir a esse"
- **W** para "duplicar o cartão a seguir ao próximo"

Este jogo representa-se então compactamente da seguinte forma.

$$\mathbf{T}(\mathbf{fT})\mathbf{AW} \longrightarrow \mathbf{A}(\mathbf{fT})\mathbf{W} \longrightarrow \mathbf{fT}(\mathbf{W}(\mathbf{fT})) \longrightarrow \mathbf{W}(\mathbf{fT})$$

Podemos ainda simplificar mais, tornando a definição das letras mais precisa matematicamente. Uma vez que cada instrução só afecta um número finito e conhecido de cartões a seguir, podemos representá-los por variáveis e descrever o efeito de cada letra usando essas variáveis. Por exemplo: a instrução "trocar os dois cartões seguintes" pode ser descrita como transformando xy em yx; uma vez que esta instrução é representada pela letra **T**, podemos escrever compactamente

$$\mathbf{T}xy \longrightarrow yx\,.$$

De forma semelhante, sugerimos ao leitor que verifique as definições seguintes.

$$\begin{array}{ccc} \mathbf{K}xy \longrightarrow x & \mathbf{M}x \longrightarrow xx & \mathbf{P}xy \longrightarrow xyx \\ \mathbf{f}xy \longrightarrow y & \mathbf{A}xy \longrightarrow x(yx) & \mathbf{W}xy \longrightarrow xyy \end{array}$$

Observe-se que para definir a letra **f** precisamos de referir os dois cartões seguintes para garantir que a pilha não fica vazia.

Ao jogo dos cartões jogado com sequências de letras é tradição chamar *Cálculo Combinatório* ou *Cálculo de Combinadores*. As letras dizem-se combinadores; as

letras **T**, **M** e **K** usadas acima estão firmemente estabelecidas há perto de cem anos como correspondendo aos combinadores que definimos.

A *ordem* dum combinador é o número de variáveis de que necessita para ser definido; assim, o combinador **M** é de primeira ordem, enquanto todos os outros que apresentámos acima são de segunda ordem. Mais adiante encontraremos combinadores de ordem mais elevada.

4. Algumas habilidades

Uma das primeiras questões que se colocam neste ponto é: de quantos combinadores é que precisamos para representar todas as operações possíveis? À partida este número pode parecer muito elevado (ou mesmo infinito); porém, se começarmos a explorar um pouco este jogo, rapidamente começamos a descobrir que é muito fácil escrever as mesmas operações de várias maneiras diferentes — pelo que complicando um pouco a sequência de instruções num cartão podemos conseguir reproduzi-lo com um pouco mais de trabalho.

Vejamos um exemplo. Suponhamos que dispomos do combinador **T**, definido acima, e ainda dos seguintes três combinadores de terceira ordem.

$$\mathbf{B}xyz \longrightarrow x(yz) \qquad \mathbf{C}xyz \longrightarrow xzy \qquad \mathbf{F}xyz \longrightarrow zyx$$

Consideremos ainda que queríamos escrever a instrução "passar o terceiro cartão da pilha para cima". Podemos defini-la como um combinador **V** tal que

$$\mathbf{V}xyz \longrightarrow zxy\,;$$

mas podemos igualmente implementar esta instrução através dos combinadores **BCT** ou **CF**:

$$\mathbf{BCT}xyz \longrightarrow \mathbf{C}(\mathbf{T}x)yz \longrightarrow \mathbf{T}xzy \longrightarrow zxy$$
$$\mathbf{CF}xyz \longrightarrow \mathbf{F}yxz \longrightarrow zxy$$

Se *definirmos* **V** como **BCT** (ou como **CF**) temos que $\mathbf{V}xyz \longrightarrow^* zxy$ — o que para efeitos práticos é perfeitamente satisfatório.

Este exemplo motiva uma regra adicional no jogo dos combinadores: uma vez que a mesma regra pode ser implementada de várias formas, permitimos substituir uma expressão A no corpo dum combinador (ou seja, não no início) por outra B desde que $A \longrightarrow^* B$. É fundamental ter em atenção que a expressão A tem de ocorrer entre parênteses — no jogo dos cartões, esta operação corresponde a substituir o texto num cartão por uma instrução equivalente. Para um combinador constituído por uma única letra, escrever **A** ou escrever (**A**) é equivalente.

Construir um combinador que implemente uma dada operação a partir de outros já conhecidos não é, à partida, trivial: implica raciocinar em sentido inverso, vendo cada expressão como resultado da aplicação de outro combinador. Para ilustrar este processo, vejamos como poderíamos proceder para encontrar um combinador **Q** tal que $\mathbf{Q}xyz \longrightarrow^* y(xz)$. Primeiro, podemos observar que a ocorrência de parênteses sugere o recurso ao combinador **B**: temos

$$\mathbf{B}yxz \longrightarrow y(xz)\,.$$

Porém, nesta expressão as variáveis x e y surgem por ordem inversa. Recorrendo ao combinador **C**, podemos retroceder mais um passo:

$$\mathbf{CB}xyz \longrightarrow \mathbf{B}yxz\,.$$

Então podemos definir **Q** como **CB**.

O leitor interessado poderá tentar usar os combinadores **B**, **C** e **T** para implementar os dois combinadores seguintes.

$$\mathbf{Q_1}xyz \longrightarrow x(zy) \qquad \text{e} \qquad \mathbf{Q_3}xyz \longrightarrow z(xy)$$

A questão de saber quantos combinadores são necessários para representar todas as instruções possíveis torna-se então mais interessante. Vimos que os combinadores **B**, **C** e **T** permitem definir muitas funções; mas é fácil perceber que não chegam para implementar o combinador **M** ou o combinador **K** que referimos anteriormente. De quantos combinadores precisamos então?

A resposta é surpreendente: apenas um. Na prática, contudo, este combinador **X** é tão pouco intuitivo[1] que é habitual trabalhar com dois: o combinador **K**, que já referimos atrás, e o combinador **S**, definido por

$$\mathbf{S}xyz \longrightarrow xz(yz)\,.$$

O conjunto formado por estes dois combinadores é o que se chama um conjunto *combinatoriamente completo*: todos os outros combinadores podem ser escritos à custa dos elementos deste conjunto. O conjunto $\{\mathbf{X}\}$ também é combinatoriamente completo. Outro conjunto combinatoriamente completo, que também tem o seu interesse, é o conjunto $\{\mathbf{B}, \mathbf{C}, \mathbf{K}, \mathbf{W}\}$.

É bastante simples perceber porque é que os combinadores **S** e **K** formam um conjunto combinatoriamente completo: basta aplicar o tipo de raciocínio regressivo que acima exemplificámos à expressão da operação que queremos implementar. É conveniente começar por definir um combinador auxiliar: o combinador identidade **I**, tal que $\mathbf{I}x \longrightarrow x$. Podemos tomar **I** como sendo **SKK**:

$$\mathbf{SKK}x \longrightarrow \mathbf{K}x(\mathbf{K}x) \longrightarrow x\,.$$

(O combinador **SKS** também funcionaria; porém, a escolha acima é a tradicionalmente adoptada.)

Vejamos a título de exemplo como podemos implementar o combinador **M**. Queremos ter $\mathbf{M}x \longrightarrow xx$; uma vez que $\mathbf{I}x \longrightarrow x$, podemos obter a expressão final a partir de $\mathbf{I}x(\mathbf{I}x)$. Uma vez que a variável x ocorre no final de ambas as subexpressões, podemos usar o combinador **S** para obter esta palavra, implementando **M** como **SII**.[2]

Ao processo aqui exemplificado vamos chamar eliminação da variável x (a variável x, presente na expressão xx, desapareceu na expressão **SII**). Mais formalmente, a eliminação de x na expressão E é um combinador A onde x não ocorre tal que $Ax \longrightarrow^* E$.

Se quisermos implementar combinadores de ordem superior, temos de repetir este processo, eliminando cada uma das variáveis à vez. Por exemplo, para implementar **T** temos de começar por eliminar y na expressão yx e depois eliminar x no resultado. A sequência de passos é a seguinte.

$$yx \longleftarrow \mathbf{I}y(\mathbf{K}xy) \longleftarrow \mathbf{SI}(\mathbf{K}x)y \longleftarrow \mathbf{K}(\mathbf{SI})x(\mathbf{K}x)y \longleftarrow \mathbf{S}(\mathbf{K}(\mathbf{SI}))\mathbf{K}xy$$

O combinador $\mathbf{S}(\mathbf{K}(\mathbf{SI}))\mathbf{K}$ é portanto uma implementação de **T**.

Como é que podemos então eliminar a variável x duma expressão? É simples: basta usar as seguintes quatro regras.

(1) A eliminação de x em x é **I**.
(2) A eliminação de x numa expressão A onde x não ocorre é $\mathbf{K}A$.
(3) A eliminação de x numa expressão Ax, onde x não ocorre em A, é A.
(4) A eliminação de x numa expressão MN é $\mathbf{S}M'N'$, onde M' e N' são, respectivamente, as eliminações de x em M e N.

---

[1] Para além de pouco intuitivo, não é um *combinador puro*, o que significa que não é possível descrevê-lo com a notação que temos vindo a utilizar nesta exposição.

[2] O leitor atento terá dado conta de que, de acordo com o que foi exposto, se tem $\mathbf{M}x \longrightarrow x(\mathbf{I}x)$, não sendo possível continuar. Porém, esta expressão é indistinguível de xx num sentido técnico muito preciso que não iremos detalhar nesta exposição.

O leitor pode verificar que as implementações atrás encontradas para **M** e **T** são as mesmas que se obtêm seguindo estas quatro regras. Na última secção encontram-se alguns comentários sobre esta construção, bem como uma notação alternativa que pode ser útil caso se deseje eliminar variáveis em expressões mais complexas.

5. Combinadores de ponto fixo

Para além da completude combinatorial, o Cálculo de Combinadores tem outras propriedades fascinantes. Uma delas é a existência de soluções de equações de ponto fixo. Concretamente, dada qualquer expressão E envolvendo um combinador A e variáveis $x_1, x_2, \ldots, x_n$, é possível encontrar um combinador **A** tal que $\mathbf{A}x_1x_2\ldots x_n$ se reduz à expressão E com A substituído por **A**.

Por exemplo: é possível encontrar um combinador A tal que

$$Ax \longrightarrow^* \mathbf{S}(\mathbf{K}A)x(AA) \tag{1}$$

ou um combinador B tal que

$$Bxyz \longrightarrow^* B\mathbf{S}(xy)(\mathbf{S}B(\mathbf{K}B)yz)zx\,. \tag{2}$$

Este resultado pode parecer à primeira vista surpreendente; contudo, o leitor diligente já terá encontrado uma situação semelhante num dos exercícios que deixámos atrás.

A técnica geral para resolver estas equações, que é simultaneamente a demonstração deste resultado, depende da existência dum *combinador de ponto fixo*: um combinador Y tal que $Yx \longrightarrow^* x(Yx)$. Existem vários combinadores de ponto fixo (na realidade, um número infinito); um dos mais conhecidos é o combinador **UU**, em que **U** (também chamado *combinador de Turing*) é definido por

$$\mathbf{U}xy = y(xxy)\,.$$

De facto, temos directamente que $\mathbf{UU}x \longrightarrow x(\mathbf{UU}x)$, donde **UU** é um combinador de ponto fixo.

Para resolver a equação (1), começamos por encontrar um combinador $\mathbf{A_1}$ que implemente a expressão da direita, eliminando as variáveis A e x que nela ocorrem, ou seja, tal que.

$$\mathbf{A_1}Ax = \mathbf{S}(\mathbf{K}A)x(AA)\,.$$

Qualquer ponto fixo de $\mathbf{A_1}$ é solução:

$$Y\mathbf{A_1}x \longrightarrow^* \mathbf{A_1}(Y\mathbf{A_1})x \longrightarrow^* \mathbf{S}(\mathbf{K}(Y\mathbf{A_1}))x((Y\mathbf{A_1})(Y\mathbf{A_1}))$$

donde $Y\mathbf{A_1}$ é solução da equação (1).

A resolução da equação (2) é semelhante, sendo necessário eliminar da expressão do lado direito as variáveis B, x, y e z, obtendo um combinador **B**; uma solução é então $Y\mathbf{B}$, com Y um qualquer combinador de ponto fixo.

6. Números e contas

Em tudo o que vimos até aqui, a única coisa que nunca apareceu foram números. Pode, portanto, ser inesperada a afirmação de que o Cálculo de Combinadores permite fazer contas — não apenas contas simples, mas na realidade quaisquer cálculos que possam ser mecanizados. O mecanismo é simples de entender — e uma óptima abordagem à forma como um computador funciona ao nível mais elementar.

A forma como habitualmente escrevemos e trabalhamos com números é uma simples convenção. Para fazer contas no Cálculo de Combinadores, precisamos de estabelecer novas convenções.

Em primeiro lugar, vamos escolher combinadores para representar cada número. Vários autores propuseram diferentes formas de o fazer; a que vamos apresentar aqui

é devida a Henk Barendregt. Vamos denotar por $\ulcorner n \urcorner$ o combinador correspondente ao número n; temos então que

$$\ulcorner 0 \urcorner \equiv \mathbf{I} \qquad \ulcorner n+1 \urcorner \equiv \mathbf{V}(\mathbf{KI})\ulcorner n \urcorner$$

onde $\equiv$ significa "é definido por" e $\mathbf{V}$ foi definido atrás.

Dizemos que um combinador F representa uma função $f : \mathbb{N}^n \to \mathbb{N}$ se se tiver a relação

$$F\ulcorner x_1 \urcorner\ulcorner x_2 \urcorner \ldots \ulcorner x_n \urcorner \longrightarrow^* \ulcorner f\,(x_1, x_2, \ldots, x_n) \urcorner$$

para todos os valores de $x_1, x_2, \ldots, x_n$. Ou seja: o valor de f num ponto $x_1, x_2, \ldots, x_n$ pode ser calculado aplicando o combinador F a $\ulcorner x_1 \urcorner, \ulcorner x_2 \urcorner, \ldots, \ulcorner x_n \urcorner$.

Por exemplo, a função sucessor (que a cada número x associa $x+1$) é representada pelo combinador σ definido como $\mathbf{V}(\mathbf{KI})$, já que por definição

$$\mathbf{V}(\mathbf{KI})\ulcorner n \urcorner = \ulcorner n+1 \urcorner .$$

Já a função identicamente nula (que transforma cada número x em 0) é representada pelo combinador $\mathbf{KI}$:

$$\mathbf{KI}\ulcorner n \urcorner \longrightarrow \mathbf{I} = \ulcorner 0 \urcorner .$$

Será que o leitor consegue encontrar um combinador que represente a função que associa a cada valor x o valor 2? E a função que soma 3 ao seu argumento?

Para finalizar, vamos mostrar como definir funções mais sofisticadas como a soma, a multiplicação ou a exponenciação. Todas elas assentam no mesmo princípio e requerem três ingredientes fundamentais: a função predecessor; o teste a zero; e o mecanismo de definição de funções por recursão.

Comecemos pela função predecessor: a função que a cada natural positivo n associa o número $n-1$. Um combinador que implementa esta função é o combinador $\mathbf{P}$ definido como $\mathbf{T}(\mathbf{KI})$, conforme se verifica facilmente.

$$\begin{aligned}\mathbf{T}(\mathbf{KI})\ulcorner n+1 \urcorner &= \mathbf{T}(\mathbf{KI})(\mathbf{V}(\mathbf{KI})\ulcorner n \urcorner) \longrightarrow \mathbf{V}(\mathbf{KI})\ulcorner n \urcorner(\mathbf{KI}) \\ &\longrightarrow \mathbf{KI}(\mathbf{KI})\ulcorner n \urcorner \longrightarrow \mathbf{I}\ulcorner n \urcorner \longrightarrow \ulcorner n \urcorner\end{aligned}$$

O teste a zero é um combinador de escolha: permite-nos definir funções por ramos, com um resultado A se o argumento for 0 e outro resultado B se o argumento for diferente de 0. Com o sistema de numeração que escolhemos, o teste a zero é um combinador $\mathbf{Z}_?$ que pode ser implementado por $\mathbf{TK}$. De facto, temos:

$$\mathbf{TK}\ulcorner 0 \urcorner AB = \mathbf{TKI}AB \longrightarrow \mathbf{IK}AB \longrightarrow \mathbf{K}AB \longrightarrow A$$

$$\begin{aligned}\mathbf{TK}\ulcorner n+1 \urcorner AB &= \mathbf{TK}(\mathbf{V}(\mathbf{KI})\ulcorner n \urcorner)AB \longrightarrow \mathbf{V}(\mathbf{KI})\ulcorner n \urcorner\mathbf{K}AB \\ &\longrightarrow \mathbf{K}(\mathbf{KI})\ulcorner n \urcorner AB \longrightarrow \mathbf{KI}AB \longrightarrow \mathbf{I}B \longrightarrow B\end{aligned}$$

O mecanismo de definição de funções por recursão é também muito simples e assenta num princípio matemático fundamental: podemos definir uma função por casos da seguinte forma.

- Dando (directamente) o seu valor no ponto 0.
- Indicando o seu valor num ponto $n+1$ à custa do seu valor no ponto n.

Por exemplo: se definíssemos uma função f através das condições

$$f(n) = \begin{cases} 3 & n = 0 \\ f(n-1)+2 & n > 0 \end{cases}$$

ou, duma forma mais algorítmica,

$$f(n) \equiv \textbf{se } (n = 0) \textbf{ então } 3 \textbf{ c.c. } (f(n-1)+2)\,,$$

lido "se $n = 0$ então o resultado é 3, caso contrário o resultado é $f(n-1)+2$", tínhamos informação suficiente para calcular o valor de f em qualquer ponto. A título de exemplo:

$$f(0) = 3 \qquad f(1) = f(0) + 2 = 5 \qquad f(3) = f(2) + 2 = f(1) + 2 + 2 = 9\,,$$

e assim sucessivamente.

Para encontrar um combinador F que implementasse esta função f, precisávamos que F satisfizesse a condição seguinte.

$$F^{\ulcorner} n^{\urcorner} \longrightarrow^* \mathbf{Z}_? {}^{\ulcorner} n^{\urcorner} {}^{\ulcorner} 3^{\urcorner} \left(\sigma \left(\sigma \left(F \left(\mathbf{P}^{\ulcorner} n^{\urcorner}\right)\right)\right)\right) . \tag{3}$$

Porquê? Analisemos a expressão da direita à luz do que dissemos atrás. O combinador $\mathbf{Z}_?$ faz um teste a zero; portanto, a expressão $F^{\ulcorner}0^{\urcorner}$ reduz-se imediatamente a $^{\ulcorner}3^{\urcorner}$. No caso de $n > 0$, o teste a zero reduz-se à expressão seguinte,

$$\sigma \left(\sigma \left(F \left(\mathbf{P}^{\ulcorner} n^{\urcorner}\right)\right)\right)$$

que se reduz ao numeral correspondente a $f(n-1)+2$.

Ora a equação (3) não é mais do que uma equação de ponto fixo disfarçada: basta substituir $^{\ulcorner}n^{\urcorner}$ por uma variável x para obtermos uma equação na forma habitual. Sabemos já que existe necessariamente um combinador $\mathbf{F}$ que é solução desta equação; esse combinador é uma implementação da função f.

Para definir a soma, produto e exponenciação só temos então que escrever estas funções sob a forma duma definição recursiva. Ora é fácil verificar que

$$\begin{aligned} n + m &= \textbf{se } (m = 0) \textbf{ então } n \textbf{ c.c. } (n + (m-1)) + 1 \\ n \times m &= \textbf{se } (m = 0) \textbf{ então } 0 \textbf{ c.c. } (n \times (m-1) + n) \\ n^m &= \textbf{se } (m = 0) \textbf{ então } 1 \textbf{ c.c. } \left(n^{m-1} \times n\right); \end{aligned}$$

a primeira, por exemplo, significa: somar n com 0 devolve simplesmente n, enquanto que a soma de n com m (maior que 0) pode ser calculada somando n com $(m-1)$ e acrescentando 1 ao resultado. As outras equações interpretam-se de forma semelhante.

Desta forma obtêm-se as seguintes equações de ponto fixo, cujas soluções $\mathbf{s}$, $\mathbf{m}$ e $\mathbf{e}$ são combinadores que implementam estas funções.

$$\begin{aligned} Sxy &= \mathbf{Z}_? yx \left(\sigma \left(Sx \left(\mathbf{P}y\right)\right)\right) \\ Mxy &= \mathbf{Z}_? y^{\ulcorner}0^{\urcorner} \left(\mathbf{s} \left(Mx \left(\mathbf{P}y\right)\right) x\right) \\ Exy &= \mathbf{Z}_? y^{\ulcorner}1^{\urcorner} \left(\mathbf{m} \left(Ex \left(\mathbf{P}y\right)\right) x\right) \end{aligned}$$

Estas funções são apenas um cheirinho de tudo o que se pode fazer com o Cálculo de Combinadores. Conforme referimos no início, este formalismo tem o potencial de calcular qualquer função programável num computador (o que não significa que todas as funções sejam implementáveis). O leitor aventureiro poderá divertir-se a tentar encontrar um combinador que verifique se um dado número é primo ou não; que encontre o n-ésimo número perfeito; ou mesmo que calcule o n-ésimo algarismo na expansão decimal de π. Tudo, recorde-se, manipulando cartões com instruções escritas...

7. Considerações finais

Historicamente, o Cálculo de Combinadores foi concebido como um modelo (abstracto) de uma linguagem de programação, muito antes de existirem sequer projectos de construir o primeiro computador. A nossa apresentação deste cálculo foi aliás orientada neste sentido: matematicamente, os combinadores são funções (transformam combinadores noutros combinadores); porém, ao introduzi-los como instruções escritas em cartões, pretendeu-se atribuir-lhes uma conotação operacional que está

ligada ao seu interesse original. Em termos teóricos, o Cálculo de Combinadores tem um interesse especial por permitir representar funções sem recurso a variáveis — algo que não é de todo comum em Matemática.

É fácil ver que podemos escrever combinadores diferentes que se comportam da mesma forma enquanto funções. Por exemplo: vimos que os combinadores **BCT** e **CF** são duas implementações diferentes do combinador **V**. Dizemos que estes combinadores são η-equivalentes (eta-equivalentes): matematicamente, definem a mesma função. O Cálculo de Combinadores é contudo mais fino do que o habitual em Matemática: estes combinadores correspondem a *algoritmos* diferentes para realizar a mesma tarefa — uma distinção subtil mas fundamental na área da Teoria da Computação.

O problema de encontrar conjuntos combinatoriamente completos está relacionado com o problema de encontrar um conjunto mínimo de instruções que permita reproduzir todo o poder computacional de um computador. Noutras aplicações do Cálculo Combinatório, nomeadamente quando se pretende modelar sistemas com recursos limitados, é útil considerar outros conjuntos de combinadores que geram programas com características específicas. Por exemplo, o conjunto $\{\mathbf{B}, \mathbf{C}, \mathbf{M}, \mathbf{I}\}$ gera todos os combinadores não cancelativos — em termos do jogo dos cartões apresentado inicialmente, correspondem a não permitir instruções que mandem retirar cartões. Já o conjunto $\{\mathbf{B}, \mathbf{C}, \mathbf{K}, \mathbf{I}\}$ gera apenas combinadores que não multiplicam recursos. A análise dos jogos obtidos a partir de cada um destes conjuntos é um tópico extremamente interessante.

O Cálculo de Combinadores não foi de forma alguma o único modelo de computação proposto no seu tempo; aliás, o período entre as duas guerras foi extremamente prolífero em protótipos do que seriam as futuras linguagens de programação. Um outro formalismo extremamente importante, que está na base das linguagens de programação da família do LISP, e que está extremamente ligado ao Cálculo de Combinadores é o Cálculo-λ. A construção que apresentámos atrás para mostrar a completude combinatorial do Cálculo Combinatório com os combinadores **S** e **K** é aliás usada também para mostrar que o Cálculo Combinatório é tão poderoso como o Cálculo-λ; neste contexto, utiliza-se a notação $\lambda^* x.M$ para denotar a eliminação de x no combinador M, e as regras de eliminação escrevem-se como

$$\begin{aligned} \lambda^* x.x &= \mathbf{I} & \qquad \lambda^* x.A &= \mathbf{K}A \\ \lambda^* x.(Ax) &= A & \qquad \lambda^* x.MN &= \mathbf{S}(\lambda^* x.M)(\lambda^* x.N) \end{aligned}$$

com as restrições indicadas atrás. Esta notação é extraordinariamente prática, pelo que é frequentemente usada até no contexto do Cálculo de Combinadores puro. As sublinguagens do Cálculo de Combinadores referidas no parágrafo anterior estão por sua vez relacionadas com subconjuntos interessantes de termos do Cálculo-λ.

Os combinadores de ponto fixo estão na base de vários mecanismos usados nas linguagens de programação funcional.

Conforme foi várias vezes referido ao longo do texto, o Cálculo de Combinadores tem todo o poder computacional de uma linguagem de programação — sendo equivalente a formalismos como o Cálculo-λ, as máquinas de Turing, as funções parciais recursivas de Kleene, ou a linguagens existentes na prática, como o C, o Java ou o Lisp. Contudo, precisamente devido a esse poder computacional, está sujeito às mesmas limitações que todos eles — aplicando-se resultados como o Teorema da Incompletude de Gödel, a Indecidibilidade do Problema da Paragem ou o Teorema de Rice. Ao descobrir-se que todos estes formalismos tinham o mesmo poder expressivo e as mesmas limitações, atingiu-se um dos primeiros marcos históricos da Teoria da Computação: a compreensão de que noção intuitiva de algoritmo é capturável por qualquer deles, sendo todos modelos teóricos igualmente válidos

para a análise dos limites da programação. Se o leitor se quiser dar ao trabalho, poderá tentar resolver o problema $P \neq NP$ recorrendo unicamente a cartões com instruções — mas talvez não seja a melhor opção...

Bibliografia

O Cálculo de Combinadores e, de uma forma mais geral, a Lógica Combinatória, é um mundo dentro da Teoria da Computação. Uma abordagem informal ao tópico, que contudo ilustra perfeitamente a riqueza do tema, pode ser encontrada na segunda parte do fabuloso livro de Smullyan *To Mock a Mockingbird* [4], em que os combinadores são apresentados como pássaros que cantam.

Para uma abordagem mais séria (e matematicamente mais dura), a referência absoluta são os capítulos 7 e 9 do livro *The Lambda Calculus*, de Henk Barendregt [1]. Embora o tópico dominante deste livro seja, como o nome indica, o Cálculo-λ, estes dois capítulos dedicam-se com algum detalhe à Lógica Combinatória. O Capítulo 5 do mesmo livro apresenta uma visão algébrica alternativa do Cálculo de Combinadores.

Várias das questões mais gerais aqui afloradas são ainda discutidas noutros artigos da colectânea. O leitor interessado em questões mais gerais de computabilidade (o que é que se pode e não pode fazer com um computador — e em particular com o Cálculo de Combinadores) encontrará uma interessante e abrangente exposição dos resultados mais relevantes no artigo de Luís Gil [2].

O artigo de Bruno Montalto [3] discute detalhadamente o problema $P \neq NP$, um dos problemas em aberto mais importantes da Matemática contemporânea, recorrendo a outro modelo teórico de computação (a máquina de Turing), mais formal e mais "próximo" daquilo que estamos habituados a ver como um computador. É interessante para o leitor comparar as máquinas de Turing e o Cálculo de Combinadores; desafio-o a demonstrar formalmente que são equivalentes...

Referências

[1] H.P. Barendregt, *The Lambda Calculs: Its Syntax and Semantics*. Elsevier, 1984.
[2] L. Gil, *????*. Neste volume, p.???–???.
[3] B. Montalto, *????*. Neste volume, p.???–???.
[4] R. Smullyan, *To Mock a Mockingbird*. Oxford University Press, 2000.