Do conceito de fórmula ao de estrutura de informação

por F. Teixeira de Queiroz
(Instituto Gulbenkian de Ciência)

Introdução — Um matemático, ligado a problemas de informática, é normalmente solicitado para tratar problemas de dois tipos diferentes: ou lhe é pedido para efectuar um estudo dum problema concreto, bem definido, ou o que é pretendido é um utensílio para abordar uma classe de problemas, todos da mesma natureza, mas podendo apresentar uma certa diversidade.

No primeiro caso, concluído o estudo, a entidade que solicita o trabalho poderá eventualmente desejar o tratamento de um ou outro ponto de pormenor, mas, dum modo geral, não tenciona recorrer novamente ao matemático. Ao contrário desta situação, o segundo tipo de problemas que referimos, pressupõe a utilização prolongada dum ordenador. Em tais condições torna-se fundamental garantir a total independência do utilizador relativamente ao matemático que concebe o programa, mesmo que esse utilizador desconheça completamente qualquer linguagem de programação. O papel do matemático será então o de conceber e garantir a manutenção dum *sistema* susceptível de tratar a família de problemas que se deseja resolver.

Desta forma, a constituição dum sistema implica necessariamente a existência de duas entidades que se completam [1]:

a) Uma entidade que cria o sistema e assegura a sua manutenção.

b) Um utilizador ou grupo de utilizadores.

Para estes é fundamental que essa utilização seja o mais transparente possível e que, simultaneamente, seja suficientemente bem adaptada à resolução dos problemas que deva tratar. Um dos modos de alcançar este duplo objectivo consiste na organização e montagem duma linguagem não processual. Vejamos o que se entende por tal:

O matemático, ao redigir um programa, utiliza usualmente sequências de instruções pre-construídas, e já testadas, que mantém em biblioteca. Essas sequências estão organizadas em unidades mais ou menos autónomas, *os processos*, e executam algorítmos que realizam tarefas específicas. As linguagens de programação têm meios que permitem activar esses processos e fornecerem-lhes os dados que eles deverão trabalhar. Para a utilização de tais processos torna-se necessário o conhecimento quer da linguagem em que os mesmos estão ridigidos quer do modo de os chamar.

Dizemos que uma linguagem que permite a chamada de processos é processual.

A fim de evitar ao utilizador a tarefa de estudar uma linguagem deste tipo, tem o matemático a possibilidade de construir uma linguagem não processual. Esta, apoiar-se á num programa redigido numa linguagem de programação e será constituída por um pequeno número de vocábulos chave que poderão ser escolhidos de combinação entre o matemático e o utilizador. Tais vocábulos, ao serem usados como dados, permitirão a

realização de tarefas complexas podendo conter, inclusivamente, a chamada de processos. Eles serão assim usados pelo utilizador duma forma indirecta, libertando-o completamente do conhecimento da linguagem em que foram redigidos. (Esta será a metalinguagem da linguagem criada).

A criação duma linguagem não processual põe em primeiro lugar a necessidade duma análise dos objectos que terão de ser manipulados para que possa ser escolhida uma representação conveniente dos mesmos. Determinada a natureza desses objectos, ou seja a *estrutura da informação* contida neles, importa seguidamente, redigir os processos input-output, isto é, as partes do programa que convertem os elementos que o utilizador deseja tratar, na representação que os mesmos deverão ter quando arquivados pelo ordenador.

Seguidamente haverá que estudar os algoritmos que manipulam essas estruturas.

Pensamos que o exemplo que damos seguidamente poderá ter interesse, já que no caso tratado, o potencial utilizador a que nos referimos anteriormente, é o matemático. Com efeito, a linguagem não processual que foi montada destina-se à manipulação de fórmulas, tendo sido usada como metalinguagem o ALGOL [2, 3].

1 — No seu trabalho, o matemático utiliza expressões analíticas tais como

$a)$ $(a-1)\cdot\cos x+(a+1)\sin(\mathrm{PI}/2-x)$

$b)$ $\dfrac{\partial^2\psi}{\partial x^2}-\dfrac{1}{c^2}\cdot\dfrac{\partial^2\psi}{\partial t^2}$

$c)$ $\sin^4 x-\sin 4\cdot x$

$d)$ $\dfrac{d}{dt}\left(\varphi(t)\cdot\dfrac{dy}{dt}\right)+\psi(t)\cdot y$

etc., que transforma desenvolvendo, simplificando, ou substituindo variáveis por expressões. Cada uma destas expressões fornece um conjunto de informações que permite a sua utilização, de acordo com regras bem definidas, mas de um modo heurístico.

O objectivo da construção dum manipulador de fórmulas consiste na mecanização de toda essa actividade com vista à sua execução por um ordenador.

De acordo com o programa assinalado no início, teremos como primeira tarefa, que determinar a estrutura de informação contida nas fórmulas referidas ou noutras do mesmo tipo.

Uma primeira análise mostra-nos que elas contêm sinais de operação, cada um dos quais liga entre si dois operandos para dar origem a uma operação algébrica. O resultado desta poderá por sua vez constituir um operando de nova operação. De tal facto resulta um caracter recursivo para a linguagem utilizada pelo matemático e a necessidade de associar a ela estruturas de informação da mesma natureza. Dado que cada sinal operatório liga dois operandos, torna-se possível representar uma fórmula por meio duma arborescência binária. A cada nodo desta estará ligado um sinal operatório e a cada ramo um operando.

Para vermos melhor a conexão existente entre fórmulas e arborescências binárias, consideremos a seguinte definição recursiva destas últimas:

Por arborescência binária entende-se um objecto que ou se reduz a um único elemento (folha) ou é o enlace de duas arborescências binárias. O ponto que faz o enlace dessas arborescências é um nodo a que se dá o nome de raiz.

Aparentemente esta definição é uma tautologia. Para vermos que não o é, deverá introduzir-se o conceito de altura da arborescência: dizemos que uma arborescência é de altura um se se reduzir a um único ponto, e dizemos que é de altura não superior a i se for o resultado do enlace de duas

arborescências de alturas não superiores a $i-1$.

Vemos que a definição de arborescência que demos é na realidade uma definição por indução finita.

A associação de arborescências binárias a expressões analíticas leva-nos a considerar diversos tipos de nodos. Pelo que já dissemos, vemos que a cada sinal operatório deveremos associar um tipo de nodo. Isso leva-nos a considerar cinco tipos diferentes de nodos (Representando-se a potenciação por $X \uparrow N$). A existência de funções elementares leva-nos a introduzir um novo tipo de nodo reduzindo-se um dos ramos da arborescência à sigla da função. Finalmente, a operação de derivação dará origem a outro tipo de nodo, contendo um dos ramos apenas a variável em ordem à qual se deriva.

Num formalismo estricto, a existência de diversos tipos de nodos levar-nos-ia a considerar, associado a cada nodo, mais um ramo contendo como única informação o tipo desse nodo. Pensamos porém qua tal prática complicaria inutilmente a descrição que estamos a fazer. Em vez disso admitiremos que é possível aplicar a qualquer arborescência o selector NODO ( ) que determina o objecto elementar que está contido na sua raiz. Com dois outros selectores (DIR ( ), ESQ ( )) poderemos localizar os nodos consecutivos, da arborescência (direito e esquerdo) caso esta não se reduza a um ponto.

Com estes tres selectores é possível criar um processo, chamemos-lhe ANALISAR, que determina a natureza da raiz e localiza os dois nodos consecutivos de qualquer arborescência. Terá um papel inverso de um outro, a que darei o nome de COLECTOR DE ENLACE, que permite enlaçar duas arborescências a partir dum nodo. Estes dois processos constituem o mecanismo fundamental de qualquer sistema que tenha que lidar com arborescências binárias. Eles permitirão não só analisar como também criar novas arborescências, de acordo com a natureza da raiz.

Um exemplo típico de aplicação destes processos trata-se da maneira de organizar formas de percorrer a arborescência. Entendemos por tal a determinação escalonada de toda a informação contida nela.

No algoritmo que damos em seguida, os pontos terminais são assinalados pelo anulamento do seu ramo esquerdo, estando a informação contida no nodo e no seu ramo direito. A possibilidade de editar essa informação está contida no processo INFORMACAO. Quanto ao processo EDITAR edita apenas o tipo da raiz.

O algorítmo que está redigido em ALGOL, e utiliza a natureza recursiva do conceito de de arborescência será

```
«PROCEDURE» PERCORRER (ARB);
«VALUE» ARB; «INTINGER» ARB;
«BEGIN»«INTEGER» NODO, DIR, ESQ;
ANALISAR (ARB, NODO, DIR, ESQ);
«IF» ESQ = 0 «THEN»
INFORMACAO (NODO, DIR) «ELSE»
«BEGIN» EDITAR (NODO);
        PERCORRER (ESQ);
        PERCORRER (DIR); «END»
«END» PERCORRER;
```

Obtemos processos análogos a este se intercalarmos a edição do nodo entre o percurso do ramo direito e o do ramo esquerdo ou se colocarmos essa edição posteriormente ao percurso dos dois ramos.

Estando a arborescência associada a uma fórmula, o primeiro tipo de percurso origina a representação da fórmula em notação polaca, o segundo tipo gera uma notação sensivelmente análoga à usual e, finalmente, o terceiro método dá origem à representação da fórmula em cadeia polaca inversa.

No caso do percurso ser feito com a edição do nodo colocada antes do percurso dos dois

ramos da arborescência dizemos que é feito em pré-ordem. Quando a mesma edição é colocada entre o percurso dos dois ramos da arborescência damos o nome de percurso em pós-ordem. (Isso é devido à importância que quase sempre tem o ramo esquerdo da arborescência. Assim no algoritmo que damos, é aí que está a indicação de se ter obtido um ponto terminal). Também se usa neste caso, por vezes, o nome de percurso simétrico. Finalmente, quando a edição do nodo se coloca no final do percurso este recebe o nome de percurso em ordem final.

Qualquer que seja a ordem pela qual o percurso é feito, obter-se-á sempre uma lista de símbolos. Será uma imagem da arborescência binária.

Na figura representamos a arborescência que corresponde à expressão

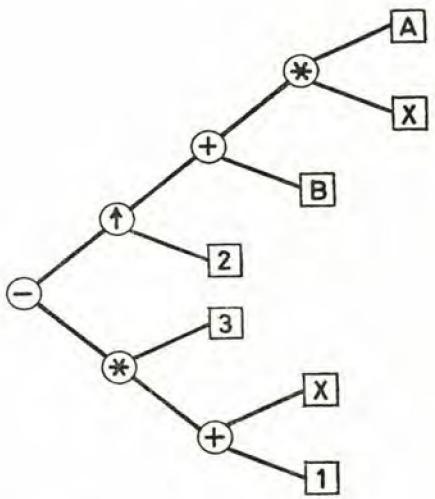
$$(a \cdot x + b)^2 - 3 \cdot (x + 1)$$

bem assim como a cadeia de símbolos que resulta do seu percurso segundo as ordens definidas. Vemos que, realmente, os percursos em pré-ordem e em ordem final geram respectivamente uma cadeia polaca e uma cadeia polaca inversa.

Pelo que acabamos de dizer, construir linguagens que manipulam fórmulas, implica criar um conjunto de algorítmos que operam sobre arborescências binárias e ainda algorítmos que convertam fórmulas em arborescências e arborescências em fórmulas.

2 — A nosso ver, o exemplo que demos, ilustra bastante bem a necessidade de determinar a estrutura que suporta um determinado tipo de informação, sempre que se torne necessário um tratamento eficiente da mesma ou sempre que se tenha que lidar com objectos de natureza complexa.

Um outro aspecto que o exemplo dado pode também mostrar é a possibilidade de usar representações distintas para a estrutura da informação e para a sua representação interna num ordenador. Alguns manipuladores de fórmulas representam-nas por meio de cadeias polacas, as quais na realidade são cadeias de símbolos (listas).



EXPRESSÃO A PERCORRER

(A*X+B)↑2−3*(X+1)

EM PRÈ-ORDEM (notação polaca)

−↑+*AXB2*3+X1

EM ORDEM SIMÉTRICA

A*X+B↑2−3*X+1

EM ORD. FINAL (notação polaca inversa)

AX*B+2↑3X1+*−

Pensamos ter mostrado a necessidade duma classificação e do estudo sistemático dos diferentes tipos de estrutura que poderão aparecer. Numa tal classificação o ponto importante é analisar as interdependências existentes entre os diversos elementos de cada objecto da família a estudar, por forma a verificar se existem hierarquias, priori-

dade, referências cruzadas, etc. Importa seguidamente determinar a classificação dos diferentes tipos de objectos da forma mais «pobre» possível, isto é, classificar os diversos objectos no menor número de famílias por forma a permitir o seu tratamento eficiente em ordenador.

Exemplifiquemos:

Uma árvore geneológica dá-nos um exemplo dum tipo de hierarquia muito comum em estruturas.

A notação de BACKUS, usada por exemplo, para descrever a linguagem ALGOL, apoia-se na propriedade de qualquer expressão admissível da linguagem ser quer um elemento simples quer o enlace de duas expressões admissíveis [4].

A língua portuguesa fornece nos um bom exemplo sobre prioridades: as palavras, que são listas de letras. ordenam-se no dicionário por ordem lexicográfica de forma a permitir a fácil localização dum vocábulo. Cada palavra fica classificada pela sua primeira letra o que origina a criação de vinte e quatro classes. Porém, para um tratamento eficiente em ordenador, será possivelmente preferível classificar os vocábulos pela sua letra final pois isso determina um menor número de classes. Além disso a sua parte terminal poderá dar imediatamente informações sobre a natureza do vocábulo com que se lida.

3 — Certamente que não nos é possível esgotar os exemplos que apontam para uma individualização do conceito de estrutura de informação e para a necessidade do seu estudo [5, 6, 7, 8].

A construção duma linguagem que opera sobre uma dada família de objectos põe, tal como acontece com as linguagens naturais, um certo número de problemas de semiótica. Esta divide-se usualmente em sintaxe, semântica e pragmática.

A descrição formal duma linguagem, usando por exemplo a notação de BACKUS, permite resolver duma forma clara o problema da sintaxe. Por meio dela poderemos saber se uma instrução foi redigida com uma forma correcta ou não.

São os problemas de semântica e de pragmática que poderão criar dificuldades entre o criador duma linguagem e o utente da mesma.

O valor semântico dos vocábulos, isto é, o significado das palavras no contexto da frase, terá de ser caracterizado duma forma não ambígua de maneira ao utilizador e ao construtor usarem de maneira idêntica as palavras chave que caracterizam a linguagem. A forma mais cómoda de caracterizar o significado dos vocábulos é por meio dum manual de utilização da linguagem. Parece-nos que as tentativas de formalização da semântica, por muitos méritos que tenham, não são de molde a facilitarem a sua utilização prática. Veja-se, por exemplo, o cálculo λ.

Não desejamos terminar este artigo sem chamar a atenção do leitor que por ventura se meta a construir uma linguagem, para os problemas que envolvem o outro aspecto da semiótica: a pragmática.

A pragmática estuda as relações duma linguagem com o seu utilizador. Ora enquanto o construtor duma linguagem e o seu utilizador têm que ter visões idênticas no que diz respeito à sintaxe e à semântica, já o mesmo não se pode dizer no respeitante à pragmática. Enquanto o primeiro pensa em termos de instruções e de chamadas a processos usando uma linguagem de programação (Portanto em termos duma metalinguagem), o último considera-a em termos de utilização. A coordenação das duas prespectivas é fundamental para que a linguagem possa ser eficaz.

BIBLIOGRAFIA

[1] Denis, J. B., *The Design and Constructions of Software Systems*, in Advanced Course on Software Enginering. Springer.

[2] Riet, R. P. van, *Formula Manipulation in ALGOL 60*, Mathematical Centre Tracts, Amsterdam

[3] Queiroz, F. T., *Manual para Manipulação de Fórmulas*, Instituto Gulbenkian de Ciência.

[4] Naur, P. e outros, *Relatório revisto sobre a linguagem algoritmica ALGOL-60*. Tradução de Gaspar Teixeira em «Gazeta de Matemática» n.os 91-95 e seguintes.

[5] Knuth, D., *The Art of Programing*, Vol. 1, 2, 3 Addison Wesley.

[6] D'Império, N., *Data Structures and Their Representation in Storage*. Anual Review in Automatic Programing. n.º 5.

[7] ———, *Machine Intelligence*. Edimburg University Press.

[8] Wegner, P., *Progranig Languages, Information Structures and Machine Organization*, McGraw-Hill Series in Computen Scienco.