Abstração, em Matemática, pode ter consequências práticas

Praciano-Pereira, T. *

3 de outubro de 2023
preprints da Sobral Matemática
no. 2023.04
Editor Tarcisio Praciano-Pereira
tarcisio@sobralmatematica.org

Resumo

Estou lidando com um conceito matemático, abstração que tem uma diferença significativa com a mesma palavra no vocabulário da língua comum, em português, por exemplo. Um grupo é uma abstração matemática e suas propriedades fazem parte da abstração é o caso da lei da distributividade da soma relativamente ao produto. Estou dando um exemplo para mostrar como a lei da distributividade pode economizar muito dinheiro. Esta lei abstrata tem um uso prático.

palavras chave:abstração,programa de computador, propriedade distributiva

I am dealing with a mathematical concept, abstraction, with a significant difference with the same word in the vocabulary of common language, in English, for example. A group is a mathematical abstraction and its properties are part of the abstraction, for an instance he distribitivity law of sum relatively to the product. I am giving you an example to show how the distributivity law of sum relatively to the product can save you a lot of money. This abstract law has a practical use.

keywords: abstraction, computer program, distributivity law.

1 Que é abstração

A palavra *abstração*, em Matemática, assim como em Computação, tem um sentido diferente daquele da linguagem usual e é preciso tratar desta diferença porque ela atrapalha.

Enquanto que na linguagem usual quer dizer "*analisar isoladamente um aspecto, contido num todo, sem ter em consideração sua relação com a realidade*", uma definição que fui buscar no *dicionário*, em Matemática significa *criar um modelo que amplie a realidade dum objeto de modo que ele possa ser usado com maior diversidade.*

Exemplo 1 (Abstração) *em Matemática*

Vou dar um exemplo, o par $(\mathbf{Z}, +)$ *do conjunto dos números inteiros com a operação de adição tem as seguintes propriedades*

1. *Em* $\mathbf{Z}$ *existe um elemento, chamado* elemento neutro, *o zero,* 0*, tal que para qualquer inteiro* a *é verdade que* $a + 0 = 0 + a = a$.

*tarcisio@sobralmatematica.org

2. *Para todo elemento $a \in \mathbf{Z}$ existe um elemento designado pelo símbolo $-a$ tal que $a + (-a) = -a + a = 0$, que é chamado de* inverso aditivo *de a.*

3. *Dados quaisquer dois elementos de $a, b \in \mathbf{Z}$ é verdade que $a+b = b+a$, a propriedade comutativa da adição em $\mathbf{Z}$.*

4. *Dados quaisquer dois elementos de $a, b, c \in \mathbf{Z}$ é verdade que $a + (b + c) = (a + b) + c$, a propriedade associativa da adição em $\mathbf{Z}$.*

*Se você considerar o conjunto $\mathcal{H}$ das horas dum relógio, e a operação "*soma de horas*", as quatro propriedades enumeradas acima valem para o par $(\mathcal{H}, +)$ em que estou usando o mesmo símbolo da adição de números inteiros para representar a* soma de horas.

Desta forma se símbolo (G, o) em que G é um conjunto e "o" é uma operação definida em G gozar das mesmas propriedades listadas acima para $(\mathbf{Z}, +)$ eu vou dizer que (G, o) é um grupo comutativo*, e eu criei* modelo *e grande parte das propriedades dos números inteiros valem para os elementos de (G, o) como é o caso com $(\mathcal{H}, +)$.*

Como aplicação*, e uma das inquietudes que temos quando um modelo é criado é saber se ele tem* aplicações*, eu posso agora resolver a equação*

$$x + 7 = 3;\ x, 7, 3 \in \mathcal{H}; \tag{1}$$

usando exatamente as mesma técnicas que usaria para resolver $x + 7 = 3$ no conjunto dos números inteiros. Basta copiar a solução da equação em $\mathbf{Z}$ e aplicá-la em $\mathcal{H}$ para obter o resultado. É preciso ter apenas o cuidado de observar quem é o inverso aditivo em $(\mathcal{H}, +)$ de 7 que é 5, porque $7 + 5 = 12$ e 12 é o elemento neutro *da adição em $(\mathcal{H}, +)$.*

$$x + 7 = 3;\ x, 7, 3 \in \mathcal{H}; \tag{2}$$
$$x + 7 = 3 \ \Rightarrow\ (x + 7) + 5 = 3 + 5; \textit{existência do inverso}; \tag{3}$$
$$(x + 7) + 5 = x + (7 + 5); \textit{pela propriedade associativa}; \tag{4}$$
$$(x + 7) + 5 = x + (7 + 5) = x + 12 = x; \textit{existência do neutro}; \tag{5}$$
$$x + (7 + 5) = 3 + 5 = 8 \ \Rightarrow\ (x + 7) + 5 = x + (7 + 5) \ \Rightarrow\ x = 8; \tag{6}$$
$$8 + 7 = 3 \textit{ a solução da equação!}; \tag{7}$$

Na sequência de expressões nas equações (eq.2) (eq.7) eu fiz uso das propriedades da estrutura de grupo comutativo *que o conjunto das horas tem com a operação* adição de horas *e você pode observar que em nenhum momento eu passei algum número para o outro lado trocando o sinal, uma das fontes de erro mais comum na Aritmética.*

Este modelo se chama grupo comutativo *porque há grupos em que a* propriedade comutativa *falha. Abra um livro de Álgebra e você vai encontrar nas primeiras páginas a* estrutura de grupo *junto com vários exemplos bem arbitrários como o conjunto das posições dum triângulo no plano junto com a operação de rotação.*

A *estrutura de grupo comutativo* é uma abstração do par $(\mathbf{Z}, +)$, quer dizer faz a abstração das propriedades que este par tem para aplicá-las em diversas situações diferentes, como por exemplo resolver a equação no conjunto das horas.

De forma exatamente análoga eu posso encontrar qual é a rotação que aplicada a um determinado triângulo vai produzir o mesmo triângulo em outra posição, ou posso aplicar os mesmos passos para resolver qualquer equação num conjunto arbitrário (G, o) que eu tenha provado que é um *grupo*.

A estrutura de *grupo* é uma *abstração* da estrutura $(\mathbf{Z}, +)$. A palavra *abstração* tem quase que o sentido oposto, dentro da Matemática, ou da Computação que seu sentido usual na linguagem, está *fortemente ligada à realidade*!

Por volta de 1945, dois matemáticos, Eilenberg e MacLane escreveram um artigo em que inventaram a *teoria das Categorias* que é uma teoria geral de modelos para Matemática, nem eles mesmos estavam seguros que estavam abrindo uma avenida dentro da Matemática e da Computação a ponto de chamarem a teoria deles de *general abstract nonsense*, uma "*teoria geral da estupidez*" e é como ela é *carinhosamente* conhecida entre os matemáticos.

- Um *objeto*, em Computação, corresponde para à Matemática uma *estrutura algébrica*
- uma *classe*, em Computação, corresponde para à Matemática uma *categoria*.

Para um matemático entender *programação orientada a objetos* é como cortar manteira com faca! Eu rapidamente aprendi *programação orientada a objetos*.

É disto que estou tratando neste artigo, *ganhamos com a abstração* e eu entendo que não é difícil introduzi-la no ensino de Matemática e deve ser feito logo no começo para aproveitar a capacidade imensa que têm os jovens de adquirir o novo, aliás com o defeito *do novo pelo novo*.

2 Distributividade do produto relativamente à adição

Aqui está um exemplo de como *abstração* é uma *questão prática* que pode lhe economizar *bilhões de reais*. Somente espero que você não pretenda usar a *abstração* de forma tão pragmática...

O meu grande temor é que algum banqueiro pretenda me contratar para fazer economias nos sistemas bancários coisa que certamente implicaria na despedida de funcionários, aumento do desemprego e, mais terrível, aumento dos lucros de *elementos totalmente in[uteis para a sociedade, os banqueiros*.

Certamente você sentiu que é existe *muita abstração* na explanação, caracterizando as propriedades das operações numa série de itens que se assemelham à *burocracia judiciária*...

Este é o método axiomático de descrever a ciência, ele tem alguns defeitos, mas tem suas vantagens enquanto que a *burocracia judiciária* serve apenas para humilhar a população e dar alento aos *inúteis banqueiros* que se servem dela.

Eu vou lhe dar dois exemplos de definição duma função em `python3` em que numa delas, `F2`, estou simplificando a expressão que define a outra função `F1`, com o uso da *propriedade distributiva da multiplicação relativamente à adição*.

São exemplos que saem do meu projeto de reengenharia do Cálculo, [Pra19a], em que estou usando a linguagem de processamento `python3` para construir exemplos de conceitos do livro.

Eu defini as duas funções abaixo que coloquei num arquivo denominado `Abstracao.py` que vai ser chamado pelo `python3`. Você tem uma cópia deste arquivo algumas linhas abaixo, e em seguida vou mostrar-lhe como fazer uso dele, no meu sistema `Debian/gnu/linux`. Lamento não poder dar-lhe exemplos em outro "sistema" porque somente tenho acesso a computadores e notebook rodando `Debian/gnu/linux`.

Exemplo 2 (duas) *funções*

Na verdade não são duas funções e sim duas expressões que definem a mesma função.

O arquivo `Abstracao.py`

```
## arquivo Abstracao.py
def F1(n,a,b): return a*pow(n,2)/2 + n*(2*b-a)/2.0;
def F2(n,a,b): return n*(a*(n-1) + 2*b )/2.0;
for k in range(10000): print(F1(k,2,3));
## for k in range(10000): print(F2(k,2,3));
```

Na definição de `F2` eu usei a *propriedade distributiva da multiplicação relativamente à adição* e na definição de `F1` eu não usei esta propriedade.

Nos dois casos eu defini uma função da variável n usando os dois parâmetros a, b. Para "*chamar*" esta função é preciso passar-lhe n, a, b.

Vou lhe dar um exemplo de uso da mesma que você pode repetir e até verificar a diferença de tempo de processamento entre ambas.

3 Verificando o tempo de processamento

Em `bash`, numa distribuição `Gnu/Linux` tem uma função, `time`, que nos permite verificar o tempo de processamento dum programa, e eu vou lhe dar um exemplo de uso neste caso. Você irá verificar que o uso da *propriedade distributiva da multiplicação relativamente à adição* produz uma economia de tempo.

Ao definir as duas funções no exemplo, na segunda expressão eu usei a distributividade para reduzir duas operações. Na primeira expressão há 8 operações, na segunda há 6 operações, e isto é importante, pense num programa em que $F1$, ou $F2$, esteja sendo chamada um bilhão de vezes, duma para outra há uma economia de 2 bilhões de operações!

Ninguém usa `Linux` e sim alguma linguagem interpretada que estabelece a comunicação usuário-Linux. `bash` é uma dessas linguagens frenquentemente usada em computadores rodando `Linux`.

Acho que somente este exemplo deve mostrar-lhe a importância da abstração que o método axiomático nos fornece.

Este é um exemplo de uma atividade muito importante em computação, ou em qualquer atividade em que hajam algoritmos em uso, a *otimização do algoritmo*.

Esta é uma cópia do meu arquivo `Abstracao.py`

```
def F1(n,a,b): return a*pow(n,2)/2 + n*(2*b-a)/2.0;
def F2(n,a,b): return n*(a*(n-1) + 2*b )/2.0;
for k in range(10000): print(F1(k,2,3));
## for k in range(10000): print(F2(k,2,3));
```

```
Com F1
real 0m0,041s
user 0m0,013s
sys 0m0,028s
```

```
Com F2
real 0m0,039s
user 0m0,021s
sys 0m0,018s
```

4 Explicando as saídas de dados

Deixe-me primeiro explicar-lhe como usei os programas e em seguida vou discutir as duas saídas de dados, com `F1` e com `F2`.

Primeiro eu criei o arquivo `Abstracao.py` e neste momento a chamada à função `F2` está inibida porque na frente do `for` eu coloquei o `símbolo de comentário` e consequentemente `python3` irá ignorar esta chamada e vai usar apenas chamada à função `F1`. Depois eu inverti a colocação do *símbolo de comentário* para fazer com `python3` executasse a outra função.

Eu executei

Exemplo 3 (Uso do `time`) *para medir tempo de processamento*

```
time(python3 < Abstracao.py)
```

4.1 O programa `time`

`time` é uma função do `bash` dentro da qual eu executo um `processo`, neste caso estou chamando o meu arquivo `Abstracao.py` com o comando descrito no exemplo 3 e você pode ver o tempo de processamento com três dados,

`time` é uma função do `bash` e as funções em Computação significam algo semelhante ao que *função* significa em Matemática, por exemplo o seu uso se faz com `time()` em que eu devo colocar entre os parentesis *alguma coisa*, pode ser um parâmetro ou outra função. Neste caso estou colocando um `processo`, quer dizer, uma operação computacional muito comum em sistemas do tipo `Unix` e `Linux` é um `unixoide`.. O símbolo matemática da desigualdade num comando do sistema é um `direcionador`, ele indica "*quem vai para onde*" e neste caso estou mandando o arquivo `Abstracao.py` para `python3`:

`bash` é a linguagem de comunicação usuário, `Linux`. Ninguém usa `Linux`, usa `bash` ou outro meio de comunicação com o sistema.

- `python3` é uma linguagem que o `sistema` entende como sendo um `comando do sistema`.
- Eu enviei o arquivo `Abstracao.py` como um parâmetro para `python3`.
- `time(python3 < Abstracao.py)` fez com que `time()` calculasse o tempo de processamento do `python3` sobre `Abstracao.py` e é este resultado que eu recebo do `time()`.

1. `real`,
2. do usuário, eu, `user` e
3. `sys`.

Eu não sei bem explícar-lhe a diferença entre o primeiro e o terceiro, mas o tempo do usuário costuma ser maior porque está o sistema está *perturbado* pelo uso de outros programas que estejam rodando no computador. O que me interessa para medir a rapidez dum programa são o primeiro e o terceiro itens da medição.

Usei a lista de números que vai de 1 até 10000 para testar o uso das duas funções com o que verifiquei que a função `F2`, em que usei a propriedade distributiva da multiplicação relativamente à adição, é mais efetiva. Se você rodar duas vezes seguidas este teste, os números serão diferentes, isto se deve a diversos fatores, mesmo num computador pessoal o sistema está executando diversos programas na surdina, em inglês eles dizem *in background* o que afeta a velocidade dum programa que eu esteja rodando. Então a forma segura de testar a efetividade dum algoritmo consiste em fazer um levantamento estatístico do mesmo, na prática criar um programa que chame

```
time(python3 < Abstracao.py)
```

alguns milhares de vezes jogando o resultado num arquivo e depois fazer o *cálculo de probabilidades* sobre a efetividade de cada uma das funções do algoritmo. Aqui eu simplifiquei muito o processo.

É importante observar que tudo isto foi executado com minha *inteligência natural* sem nada roubar ou plagiar de ninguém.

Índice Remissivo

referências

[Pra19a] Tarcisio Praciano-Pereira. *A reengenharia do Cálculo*. Rel. técn. Sobral Matemática, 2019.

[Pra19b] Tarcisio Praciano-Pereira. *A reengenharia do Cálculo, segunda parte*. Rel. técn. Sobral Matemática, 2019.