# Resumo do artigo *On Understanding Types,Data Abstraction, and Polymorphism*

Gustavo Lopes Rodrigues

Abril de 2021

**Resumo**

O Artigo , busca fazer uma compreensão dos termos: tipagem, abstração de dados e polimorfismo, que junto aos conhecimentos da teoria da tipagem, em busca de criar um modelo para criação de linguagens de programação forte, com alto polimorfismo, utilizando o cálculo $\lambda$. O nome desse módelo é *FUN*.

## 1 Secção 1

### 1.1 Sem tipo e com tipo

Para começar o artigo, o autor em vez de ir pela definição diretamente, ele dá exemplos de linguagens sem tipagem e o processo de como a tipagem foi naturalmente sendo incorporado.

Não tipagem significa ter uma tipagem só. Na memória do computador isso é representando por bit strings. Em LISP, são as *S-expressions*. No Cálculo lambda, são as expressões lambda. Por fim também tem os Sets em Set Theory.

A ideia de possuir tipos foi naturalmente implementada, com a necessidade de possuir diferentes formatos com diferentes usos e diferentes comportamentos. Porém, ainda é muito díficil fazer uma distinção completa entre a organização de sem tipos e de fazer realmente uma linguagem com tipagem. Um exemplo disso seria ter uma função lambda que retorna booleano ou integer.

### 1.2 Tipagem forte e fraca

Uma maneira descrever tipos é comparando com uma armadura. Armaduras protegem o usuário de danos exteriores. Em tipagem, a armadura protege os dados de serem usados de forma não desejadas ou não intencionadas. Isso acontece, pois como objetos possuem certas tipagem, ele precisa seguir as funcionalidade da tal tipagem. O problema é que mesmo assim, durante compilação, pode acontecer de essa regra

não ser obdecida e causar muitos problemas. Uma solução para esse problema é a tipagem estática.

Tipagem estática significa que variáveis de um programa são definidas explicitamentes e então checados durante tempo de compilação. Isso é importante para a checagem de erros, melhoria em performance , garante uma estrutura a ser respeitada e de fácil leitura. Porém, existe uma outra tipo de tipagem, que garante maior flexibilidade, a tipagem forte.

Tipagem forte são linguagens onde cada tipo de dado, são predefinidos como parte da linguagem

## 1.3 Tipos de polimorfismo

Monormofismo é a ideia de que todos os valores e variáveis podem ser interpretados como um único tipo, enquanto que linguagem polimorficas possuem a ideia contrária: valores e variáveis talvez tenham mais de um tipo e suas operações são aplicáveis a operandos de mais de um tipo.

Existe dois grande grupos de polimorfismo: universal e não universal(Ad-hoc).

### 1.3.1 Universal

Polimorfismo universal assume que um tipo pode assumir um infinito número de diferentes tipos.

Eis os subtipos do polimorfismo universal:

- **Polimorfismo paramétrico**: Permitir uma função ou um tipo de dados funcionar com uma gama de tipos diferentes, exibindo algum tipo de estrutura em comum(genérico)
  *Ex: templates em C++, ArrayList em Java.*
- **Polimorfismo de inclusão**: Habilidade de classificar subtipos utilizando herança
  *Ex: Extends em Java*

### 1.3.2 Não-universal(Ad-hoc)

Ad hoc é um tipo de polimorfismo no qual objetos podem ser aplicados a argumentos de diferentes tipos, porque uma objetos polimórficos podem denotar uma série de implementações distintas e potencialmente heterogêneas, dependendo do tipo de argumento(s) ao qual é aplicada.

Eis os subtipos do polimorfismo de Ad-hoc:

- **Polimorfismo de sobrecarga**: Uma estrutura funciona em diferentes tipos, e podem funcionar de maneira diferente para cada tipo. Também chamado de Overloading, este processo é puramente uma maneira sintática para usar nomes iguais para objetos diferentes. Em tempo de compilação, o compilador pode resolver a ambiguidade
  *Ex: Funções com o mesmo nome, mas que recebem parâmetros com tipos diferentes.*
- **Polimorfismo de coerção**: operação semântica necessária para converter um tipo para outro. Isso é necessário para evitar erros de compilação e as vezes o próprio compilador já faz isso.
  *Ex: soma de inteiros com números reais*

A distinção de sobrecarga e coerção pode ser um pouco confusa em algumas situações, principalmente em linguas sem tipagem e tipagem, ou até mesmo em státicas e compiladas.

### 1.3.3 Polimorfismo em monormofismo

Estas definições de polimorfismo apresentadas pelo autor são aplicáveis apenas em linguagens com uma clara noção de tipo e valor. Isso se torna um problema, pois linguagens estritamente monorficas seriam muito restritivas em seu poder de expressão. Algumas linguagens conseguem tirar algumas das restrições sobreimposta pelo padrão do monormofismo, como por exemplo: Ada e Pascal. Estas línguas extendem o conceito de monormofismo, aplicando as regras do polimorfismo atráves das maneiras que elas conseguem.

- Overloading(Sobrecarga)
- Coercion(coerção)
- Subtyping(inclusão)
- Value Sharing(paramétrico)

Lembrando apenas de que: por mais que existem paradigmas do polimorfismo presente, nenhuma dessas linguagens são "verdadeiramente polimorficas".

## 1.4 A evolução de tipos em linguagens de programação

Essa história começa com a linguagem de programação FORTRAN. Essa linguagem tinha uma propriedade interessante: diferenciar inteiros de números reais. O motivo pelo qual essa distinção era feita, é devido a necessidade de distinguir os inteiros para a realização de iteração in loops e computação em arrays.

Outras linguagems também pegaram carona nessa ideia, linguagens como o ALGOL 60 tinham também essa distinção, porém a distinção era entre inteiros, reais e booleanos. ALGOL 60 foi a primeira linguagem a ter a explícita noção de tipos e requerimentos associados em tempo de compilação para verificação de tipos.

Com o passar do tempo, as linguagens de programação foram aumentando a quantidade de tipos presentes: arrays, ponteiros, strings, char, bytes e arquivos são apenas alguns exemplos tipos que foram adicionados.

## 1.5 Sublíngua de expressão de tipos(SET)

Um dos objetivos desse artigo é de examinar os prós e contras entre riqueza e tratabilidade(acessibilidade para checagem de tipo durante compilação) para SETs.

Basicamente, existe todo um vocabulário de palavras usadas para definir os tipos em linguagens de programação, sublinguagens de expressão de tipos incluem os tipos de dados básicos, como: inteiros e booleanos.

SETs precisam ser tanto para denotaram tipo, mas também para mostrar equivalência/similaridade a quais tipos. O objetivo é que a relação entre tipos possam ser expressas e computáveis. Similaridade entre tipos pode ser referido como polimorfismo.

# 2 FUN

FUN é uma linguagem de programação baseada em cálculo lambda ($\lambda$) que junta a tipagem lambda, com capacidade de modelar polimorfismo e OOP. Esta linguagem encapsula conceitos já citados anteriormente.

Caracterísitisticas da linguagem:

- Possui tipagem e também pode funcionar sem fazer tipagem explícita.
- Tipos parametrizados
- Abstração de tipos
- Herança
- 5 tipos básicos diferentes