Resumo Sobre O Artigo “On Understanding Types, Data Abstraction, and Polymorphism”

Rafael Amauri Diniz Augusto – 651047

Outubro de 2021

Resumo

O objetivo deste artigo é prover uma compreensão breve do artigo *On Understanding Types, Data Abstraction, and Polymorphism* de Luca Cardelli e Peter Wegner. Para tal fim, pretendo falar sobre tipos de dados e tipos de polimorfismo, explorar cada um dos tipos de polimorfismo, citar suas vantagens e desvantagens e falar sobre a história do polimorfismo. Também falaremos sobre a linguagem Fun e o Cálculo Lambda.

Parte 1 - Dados

Nesta parte o autor fala sobre a história da tipagem em linguagens de programação e como a incorporação de tipagem foi desenvolvida ao longo do tempo.

Podemos pensar em um pedaço de memória como um arranjo de bits sem nenhuma regra que diz como eles podem interagir uns com os outros. Com tipagem, a forma como esses bits interagem uns com os outros é determinada por um conjunto de regras, e isso garante consistência entre essas interações. Outro motivo é padronizar quais as características de cada tipo em uma linguagem para evitar interpretações errôneas. Caso, por exemplo, uma variável de tipo float seja interpretada como sendo uma de tipo string, as consequências para a execução do programa seriam desastrosas. Logo, ter essa garantia de que a variável continuará consistente é fundamental para o funcionamento correto do programa.

Por causa disso, tipagem foi integrada às linguagens de programação para proteger o programador e aumentar sua produtividade, e isso pode ser visto bem claramente em linguagens com tipagem estática, como C, C++ e Java. Até mesmo linguagens como Python e Ruby que deixam implícita a tipagem e a escondem do programador ainda fazem todo o trabalho por detrás dos panos para proteger a aplicação e o programador.

Parte 2 – Tipos de Polimorfismo

São chamadas de monomórficas as linguagens que consideram que cada variável, uma vez declarada, deve ser exclusivamente de um tipo e interagir só com aquele tipo, e que uma determinada função só pode receber como parâmetro determinado tipo de variável. Em contraponto às linguagens monomórficas, existem as linguagens polimórficas, e dentro das próprias linguagens polimórficas existem diferentes tipos de polimorfismo.

Dentro das linguagens polimórficas o autor apresenta duas grandes estratégias: polimorfismo universal e polimorfismo *Ad-hoc*.

Dentro de polimorfismo universal existem duas principais estratégias: O polimorfismo paramétrico, por exemplo, permite que funções trabalhem com tipos próximos de dados. Já o polimorfismo por inclusion apresenta conceitos como herança, que permite que dados façam parte de várias classes ao mesmo tempo.

Já o polimorfismo *Ad-Hoc* tem duas principais estratégias: O polimorfismo *overloading* e *coercion*. Overloading é um tipo de polimorfismo que permite que funções de mesmo nome recebam parâmetros de diferentes tipos, já o polimorfismo de *coercion* permite que variáveis de tipos semelhantes possam ser passadas para a mesma função, mesmo sendo de um tipo diferente do que a função recebe. Um exemplo disso seria a linguagem C, que aceita que variáveis do tipo *short* sejam passadas para funções que só aceitam *int*, por exemplo. O autor também explora que o polimorfismo de *coercion* não se restringe a parâmetros de funções, e também pode ser aplicado a operações; Ao fazer uma soma de um númeral inteiro com um número fracionário, o resultado será um número fracionário.

Embora todos sejam tipos de polimorfismo, o autor argumenta que polimorfismo *Ad-Hoc* é um tipo de “pseudo-polimorfismo”, já que o funcionamento por debaixo dos panos tanto de *overloading* e *coercion* é que, ao invés de um valor ter vários tipos, na verdade são vários tipos sendo entendidos como sendo do mesmo valor. Por exemplo, em *overloading*, a função pode ter o mesmo nome e receber parâmetros diferentes no código fonte, mas quanto o compilador compila para código de máquina, cada função vai ter um identificador diferente. Já em *coercion*, o operador envolvido na operação precisa primeiro converter um dos operandos para outro tipo antes de realizar a operação.

Já o polimorfismo universal é entendido pelo autor como sendo “true polymorphism”, já que envolve variáveis pertencendo a vários tipos. Dentro desses, o autor argumenta que polimorfismo paramétrico é a forma mais pura de polimorfismo, pois ela permite que o mesmo objeto ou função pode ser usado uniformemente sem mudanças, *coercions* ou qualquer outra alteração em runtime. Em outras palavras, sem “adaptações”.

Parte 3 – Evolução da Tipagem nas Linguagens de Programação

Nas primeiras linguagens de programação, variáveis numéricas eram interpretadas como sendo de um único tipo. Foi a linguagem FORTRAN que desencadeou a ideia de ter dois tipo numéricos diferentes: inteiros para números sem casas decimais e reais para números com casas decimais, embora a estratégia para essa diferenciação pode ser considerada interessante – a primeira letra da variável indica se a variável será do tipo real ou inteiro.

A linguagem Algol 60 se aproveitou das vantagens dessa separação e introduziu outra forma de fazer esta distinção, dessa vez com uma *keyword* reservada para essa definição. Após ela, outras linguagens notáveis da época a usar a mesma estratégia eram Pascal, Algol 68, PL/I e Simula. PL/I em especial avançou o conceito de tipagem ainda mais ao introduzir *keywords* para diversos tipos, como arrays, ponteiros e *records*.

Além dessas citadas, o autor explora a contribuição individual de várias outras linguagens das décadas de 1960 e 1970.

Parte 4 – Cálculo Lambda e a Linguagem Fun

O autor também explora o cálculo Lambda, que é uma forma de programação onde as variáveis e funções podem suportar diversos tipos de estruturas. Por causa disso, uma função pode inclusive ser usada como uma variável dentro de si mesma, por exemplo: *funcao1(funcao1(y)).*

O autor também elabora como essa forma de interpretação consegue operar sobre diferentes tipos de objetos, e por isso é muito importante que durante a execução do programa o compilador faça todas as conversões necessárias.

Para aliviar um pouco o trabalho do compilador e consequentemente aumentar a performance do programa, surge a ideia de cálculo lambda tipado. Essa ideia tem diversas vantagens, como obfuscar menos a lógica do programa, dar um ganho de performance para o compilador e impedir o programa de rodar equações impossíveis.

Outra forma de evitar erros no código é como a língua Fun, criada pelos autores do artigo. Fun é uma linguagem de programação baseada em cálculo lambda (λ) que junta a tipagem lambda, com capacidade de modelar polimorfismo e OOP(Object Oriented Language). Esta linguagem encapsula conceitos já citados anteriormente.

Características da linguagem:

* Possui tipagem e também pode funcionar sem fazer tipagem explícita.
* Tipos parametrizados
* Abstração de tipos
* Herança
* Cinco tipos primitivos diferentes