Resumo (Abstract)

O artigo de Luca Cardelli e Peter Wegner apresenta como objetivo desenvolver a noção de um universo tipado frente a um universo não-tipado, destacando suas vantagens e motivos pelos quais deve-se dar preferência a sistemas que visam encontrar erros o mais cedo possível. Alguns dos universos não-tipados acabam por se mostrar logicamente inconsistentes, como na forma mais básica da teoria dos conjuntos matemáticos. Assim, versões tipadas são propostas visando eliminar tais inconsistências.

Na computação, os conceitos de tipagem forte e estática surgem como uma forma de se lidar com essas inconsistências. Linguagens onde todas as expressões apresentam consistência em relação aos tipos envolvidos são chamadas fortemente tipadas. De forma análoga, se um erro é detectado em tempo de compilação, temos que tal linguagem é estaticamente tipada. Se uma linguagem é dita fortemente tipada, seu compilador pode garantir que os programas irão executar sem nenhum erro de tipo.

Além das questões supracitadas, surge, junto ao conceito de um universo tipado, a ideia de polimorfismo. Polimorfismo se divide em dois subconjuntos: universal e *ad-hoc.* Ad-hoc é obtido quando certa abstração funciona, ou aparenta funcionar, sobre diferentes tipos, podendo agir de formas variadas em cada situação. Em contrapartida, o polimorfismo dito universal - também considerado como polimorfismo verdadeiro - diz respeito a uma função, que se comporta de maneira uniforme, mesmo sendo, também, aplicada a diferentes tipos.

Em termos de evolução histórica, em meados da década de 50, Fortran destacou-se por implementar, pela primeira vez, distinção entre tipos. A linguagem distingue os valores inteiros dos reais baseando-se na primeira letra do identificador. Tal distinção se mostrou necessária por dois motivos: a diferença na forma como os valores desses tipos são representados faz com que inteiros sejam computacionalmente mais econômicos e o uso destes, em iterações e operações com vetores, é logicamente diferente do que o uso de representações em ponto flutuante.

Além de Fortran, outras linguagens passaram por caminhos semelhantes em relação a necessidade de evoluir seus tipos básicos e suas operações. O Algol 60 introduziu a distinção entre tipos de forma explícita, utilizando identificadores declarativos redundantes para inteiros, reais e booleanos, sendo a primeira linguagem com grande significância a ter uma noção transparente de tipos e requerimentos associativos para checagem em tempo de compilação. Simula foi a primeira linguagem a introduzir o conceito de orientação a objetos, seguida por Smalltalk e Loops, que introduziram o conceito de ocultação de informação. Por fim, ML - uma linguagem interativa e funcional - trouxe a implementação de polimorfismo paramétrico.

Tendo em vista as caracterizações acima trabalhadas, os autores apresentaram a concepção de uma linguagem baseada em cálculo lambda. Esta linguagem, denominada Fun, complementa o cálculo lambda tipado com recursos de segunda ordem projetados para

modelar polimorfismo e orientação a objetos. Fun traz consigo a ideia de quantificação universal, que enriquece o cálculo lambda de primeira ordem, incorporando tipos parametrizados, que podem ser especializados substituindo os tipos dos parâmetros atuais por parâmetros universais, além da noção de quantificação existencial, que amplia os recursos de primeira ordem ao permitir tipos de dados abstratos, com ocultação de informações.

Concomitantemente a isso, a linguagem criada é matematicamente simples, podendo servir de base para a concepção e implementação de linguagens de programação, com construções de tipos que são mais poderosas e expressivas que as existentes. Isso se dá devido aos tipos de dados abstratos, ao polimorfismo paramétrico e as heranças múltiplas que tornam a linguagem bastante expressiva. Especificamente, ela provê uma base para a modelagem de linguagens fortemente tipadas e orientadas a objetos.

Com base nas informações provenientes do artigo, concluímos que para projetos grandes, trabalhar em um universo fortemente e estaticamente tipado é mais vantajoso, porque quanto antes se descobre um erro, mais fácil será consertá-lo e encontrá-lo em meio ao código. Se a verificação do erro for adiada, não será fácil determinar o local em que se encontra, pois analisar um código extenso não é algo trivial e demanda bastante esforço do programador. Em contrapartida, em projetos menores, trabalhar em um universo não-tipado dá ao profissional uma maior flexibilidade, uma vez que a linguagem se torna menos rígida.

Em linguagens pertencentes ao paradigma funcional, por exemplo, que adotam sistema de dados imutáveis, nos é proporcionado uma maior segurança, mesmo não se tendo certeza do tipo que vai receber. O fato destas linguagens incorporarem a ideia de não alteração de valores faz com que não exista problemas relacionados a efeitos colaterais (introduzidos devido a existência de comandos). Assim, mantém-se um nível parecido de segurança em relação ao universo tipado, dando mais flexibilidade ao programador.