Apêndice A

O paradigma funcional puro

O haskell adota o paradigma funcional puro. Isto significa dizer que tudo nele é feito através de funções, até mesmo os loops são feitos através de funções recursivas. A seguir, consta as principais características das linguagens que adotam este paradigma.

- Imutabilidade das variáveis: Uma vez que um valor é atribuído a uma variável, não é possível alterá-lo, ou seja, um termo mais correto seria “constante” e não “variável”.
- Nenhum efeito colateral: De princípio, é necessário entender que cada função, em qualquer paradigma, tem um escopo. Tendo isto em mente, uma função que tem efeitos colaterais é uma capaz de alterar dados que estão fora do seu escopo ou que permite que dados dentro de seu escopo sejam alterados por outras funções. O uso inadvertido de ponteiros e referências em C++, por exemplo, pode causar efeitos colaterais em funções que recebem estes tipos de dados como parâmetros.
- Transparência referencial: Em virtude da ausência de efeitos colaterais, é possível ter certeza que, todas as vezes que um conjunto de parâmetros *S* for passado para uma função, ela sempre retornará o mesmo resultado, o que nem sempre é verdade em linguagens que toleram efeitos colaterais.
- Tipagem forte: Algumas linguagens puramente funcionais, como o haskell, são fortemente tipadas. Isto significa dizer que as funções não aceitarão de modo algum nenhum parâmetro cujo tipo seja diferente do esperado. Isto torna o código mais rápido, uma vez que os tipos são verificados em tempo de compilação. No entanto, em haskell, caso o tipo dos parâmetros não seja especificado, a linguagem é capaz de inferir quais são os seus tipos. Isto faz com que não haja a necessidade de declarar explicitamente o tipo dos parâmetros, o que pode deixar os programas mais flexíveis, mas menos seguros.
- Legibilidade aumentada : Códigos de linguagens puramente funcionais são muito concisos, o que facilita a leitura e a compreensão deles.

Haskell e a avaliação preguiçosa

É uma técnica de avaliação de dados em que os valores das variáveis não são processados até que seja absolutamente necessário que eles sejam processados. Grande parte das linguagens mais populares hoje usam um outro sistema de avaliação, a chamada *eager evaluation*(avaliação voraz), que tem como característica o processamento de toda uma sentença/expressão assim que ela é atribuída a uma variável ou a uma estrutura de dados, não importando quando essa informação será necessária ou não. Por exemplo :

```
variavel = 2*50
```

Em uma linguagem que adota a *eager evaluation*, como C, é comum de esperar que a expressão aritmética “2*50” seja automaticamente avaliada, gerando 100, e seja atribuída a “variavel”, mas

em uma linguagem que adota a avaliação preguiçosa, a expressão aritmética só será calculada e atribuída a “variavel”, quando o seu valor for necessário. Como no caso à baixo :

```
print variavel
```

Uma vez que é necessário imprimir o valor de “variavel”, a operação aritmética “2*50” será feita e o valor 100 será atribuído a “variavel”. Abaixo, consta um exemplo um pouco mais elaborado:

```
1. a = 5*7
2. b = 8*a
3. print b
```

Tanto a variável “a”, quanto a “b” só foram avaliadas na linha 3, pois a função “print” faz com que seja necessário a avaliação de b, que por sua vez força a avaliação de “a”, já que o cálculo de “a” é necessário para o de “b”.