Pequena Introdução na Linguagem Funcional Clean

Roberta Cavalcanti de Brito

Bacharelado em Ciências da Computação 4ª fase, 2002
Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC), Brasil, 88040-900
Fone (0XX48) 331-9739, FAX (0XX48) 331-9770
roberta@inf.ufsc.br

Resumo:

Este artigo apresenta uma pequena introdução à linguagem funcional Clean mostrando seus principais conceitos e vantagens, e utilizando-se de exemplos e definições. Serão apresentados conceitos de programação funcional como polimorfismo e recursão especificados na linguagem Clean, além de características específicas desta linguagem, como o Sistema de tipagem única (Uniqueness Typing).

Palavras-chave: Funções, Sistema de Tipagem Única, Transparência referencial, Recursão, Polimorfismo.

Introdução

Programação funcional é um tipo de programação que enfatiza a avaliação de expressões ao invés de execução de comandos. As expressões nesse tipo de linguagem são formadas usando-se funções para combinar determinados valores.

A linguagem Clean, sendo uma linguagem de programação funcional, é baseada no conceito de funções matemáticas. Ela foi desenvolvida na Universidade de Nijmegen nos Países Baixos para o desenvolvimento de aplicações do mundo real (real-world applications).

Metodologia

Uma função Clean é referencialmente transparente, ou seja, seu resultado só depende dos valores de seus argumentos.Esta propriedade tem como uma importante conseqüência o fato de que, uma vez que a função foi executada com sucesso, ela sempre executará corretamente, reagindo da mesma forma, não importando o contexto em que ela se encontra.

Para desenvolver aplicações para o mundo real deve-se poder executar determinadas tarefas como, por exemplo, atualizar um banco de dados, passar uma mensagem para outro programa, atualizar um array destruindo-se uma parte dele, etc. A linguagem Clean, mesmo não sendo uma linguagem de execução de comandos, pode atualizar um objeto de forma destrutiva através de um Sistema de Tipos especial chamado “sistema de tipo único”(uniqueness typing). Ele faz com que Clean possa incorporar atualizações destrutivas de diferentes estruturas de dados (incluindo arrays) e torna possível escrever programas interativos puramente funcionais, incluindo janelas, menus,etc.

Este sistema de tipo único torna possível também o desenvolvimento de aplicações do mundo real que façam interfaceamento com sistemas não-funcionais, possibilitando assim o desenvolvimento de aplicações eficientes. Clean é a única linguagem funcional no mundo que oferecer este sistema de tipagem único.

Características

A característica chave da linguagem funcional Clean é o já comentado sistema de inferência polimórfico de tipo único (sistema de tipagem única) que possibilita um controle refinado sobe o uso de objetos;

Além disso, Clean é uma linguagem fortemente tipada baseada no esquema de tipos de

inferência Milner / Mycroft. Este inclui os tipos de alto nível, polimorfos, algébricos, abstratos, sinônimos e quantificados existencialmente.

Outra característica seria é que pode-se definir explicitamente estruturas cíclicas e compartilhadas em Clean. Devido a isto e ao fato de todos os objetos criados serem obrigatoriamente inicializados, erros em tempo de execução são raros e limitados aos 3 seguintes casos:

1. Quando funções parciais são chamadas com argumentos fora de seu domínio (ex. divisão por zero);
2. Quando Arrays são acessados com índices fora de seu alcance (ex. o array tem tamanho 20 e chama-se array[30]);
3. Quando não foi atribuído memória suficiente para uma aplicação em Clean (o tamanho do programa ultrapassa o tamanho da memória dedicada ao programa);

Definição de funções em Clean

As definições de funções em linguagens funcionais como Clean são muito similares à definição de funções na matemática. Como um exemplo, a função quadrado abaixo:

```
quadrado : : Int  -> Int
// O tipo da função: de Int para Int
quadrado n = n*n                    (1)
// O valor da função é o argumento
// multiplicado por ele mesmo
```

A primeira linha define o tipo da função: um inteiro como argumento e um inteiro como resultado. A segunda mostra como o resultado da função quadrado aplicada a um inteiro qualquer *n* deve ser computada: multiplicando o argumento por ele mesmo.

Para expressar condições pode-se definir uma função em alternativas. Para exemplificar pode-se definir a função fatorial como a seguir:

```
fat : : Int -> Int
fat 0 = 1                           (2)
fat n = n * fat (n-1)
```

Em definições de função como no exemplo acima, será usada a primeira alternativa aplicável ao argumento. Se o número a ser fatorado for 0 (zero), a função usará a linha "fat 0 = 1" (segunda linha), se não, passará para a linha seguinte não calculando o valor da linha anterior, mesmo passando por ela. Na última linha do exemplo acima podemos observar também uma ferramenta largamente utilizada na programação funcional em geral: a Recursividade. A recursividade é a chamada da função dentro dela mesma.

Todo o programa escrito na linguagem funcional Clean computa o valor da expressão Start. Provendo-se uma definição apropriada para a função Start, pode-se computar nela o valor de qualquer expressão.

O fatorial de 6 é calculado pelo seguinte programa:

```
Start : : Int
Start = fat 6                       (3)
```

O valor da expressão Start é então mostrado ao usuário (no caso é 720). Outro exemplo com a expressão Start é o famoso "Hello World", descrito pelas duas linhas a seguir:

```
Start :: String                     (4)
Start = "Hello World"
```

Para muitas funções o tipo do argumento é irrelevante. Um exemplo simples disso é a função identidade I onde seu argumento é o resultado que ela retorna. Para indicar este fato usa-se uma variável de tipo, definindo outra importante propriedade da Programação Funcional: o Polimorfismo. O exemplo abaixo mostra como se representa um polimorfismo em Clean:

```
I : : t -> t
// Tipo: de t em t (tipo t)         (5)
I x = x
```

Essas funções, que podem processar vários tipos de argumentos, são chamadas de funções polimorfas.

Conclusões

Como pode-se observar a partir dos exemplos dados, na linguagem de programação Clean, devido à transparência referencial desta linguagem, não existem efeitos colaterais. Esta transparência referencial é uma propriedade geral de Clean: o valor de uma expressão só depende das funções

definidas e de seus argumentos, que formam o termo da expressão. Isso torna possível discutir sobre esses programas usando um raciocínio equacional (como na matemática).

O sistema tipo estático da linguagem Clean garante que não ocorram erros em tempo de execução, eliminando a necessidade de testes exaustivos.

A verificação da consistência dos tipos é mais rápida e segura também, podendo-se investir mais tempo na validação e no melhoramento do comportamento do programa.

Através do sistema de tipagem única, característica exclusiva da linguagem Clean, tem-se a possibilidade de interagir com sistemas não-funcionais oferecendo acesso direto aos sistemas de arquivos e sistemas operacionais, fazer atualizações destrutivas de diferentes estruturas de dados, etc, mantendo intacta sua característica de linguagem funcional pura.

Juntas, essas propriedades tornam muito mais fácil escrever bons algoritmos, entender códigos escritos por outros e alterar funções já existentes.

Referências

1. Site “Home of Clean”: http://www.cs.kun.nl/~clean/

2. Tutorial: http://www.cs.kun.nl/~clean/About_Clean/tutorial/tutorial.html