Linguagens de Programação

Fabio Mascarenhas - 2015.2

http://www.dcc.ufrj.br/~fabiom/lp

Continuation Passing Style

- Em uma linguagem com funções de primeira classe, como *fun*, podemos expor as continuações no próprio código do programa, sem precisar mudar o interpretador

- Usamos uma transformação global chamada *continuation passing style* (CPS)

- A ideia é fazer cada expressão virar uma função que recebe sua continuação (outra função), e “retorna” seu valor chamando essa continuação

- Podemos implementar corotinas diretamente na linguagem, usando referências mutáveis

try/catch e throw com continuações

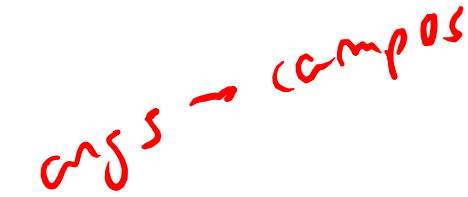
- Podemos refazer as exceções usando continuações ao invés de um valor de erro, para isso temos que manter uma “pilha de tratadores de exceção”
- Um tratador de exceção é a continuação que representa seu bloco catch e o stack pointer
- `erro` abandona a continuação atual para chamar a que está no topo da pilha, enquanto `trycatch` empilha uma continuação que executa o bloco catch e depois usa a sua continuação
- Uma corotina “herda” os tratadores de exceção no resume
- Usamos um sentinela na pilha de tratadores que “aborta” a execução

Objetos sem classes

- Um *objeto* tem duas visões: a de fora e a de dentro
- Visto de fora, um objeto é uma entidade opaca, para a qual podemos mandar *mensagens*; uma mensagem pode ter *argumentos*, que são outros objetos, e gera uma *resposta*, que também é um objeto
- Visto de dentro, um objeto tem um conjunto de *campos*, e um conjunto de métodos, que correspondem às mensagens que esses objetos podem responder
- Somente o código de um método tem acesso aos campos do objeto

Proto

- *proto* é em essência uma linguagem imperativa, como MicroC
- Só que os valores de proto agora podem ser *números* ou *objetos*
  - Objetos têm campos, que são endereços de memória
- Temos as mesmas operações de MicroC para números, mas não temos mais ponteiros
- A operação @ acessa um campo do objeto corrente




```
fun counter(n)
  object (n)
    def inc(n)
      @0 := @0 + n
    end
    def dec(n)
      @0 := @0 - n
    end
  end
end

let c = counter(0) in
  print(c.inc(4));
  print(c.dec(2))
end
```