Linguagens de Programação

Fabio Mascarenhas - 2013.1

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

Ações com *for* - aritmética

- Vamos usar a definição de ações com *for* em nosso interpretador, como na implementação do caso Soma de *eval* abaixo:

```
for {
  v1 <- e1.eval(funs)(env)
  v2 <- e2.eval(funs)(env)
} yield (v1, v2) match {
  case (NumV(n1), NumV(n2)) => NumV(n1 + n2)
  case _ => sys.error("soma precisa de dois números")
}
```

- Podemos usar *for* e recursão pra ações mais complexas, como a que avalia os argumentos para uma função

- Também podemos mudar a definição de ação sem precisar reescrever todos os casos do interpretador

Chamadas de função

- Para avaliar uma chamada de função, precisamos avaliar a expressão que dá a função, além de todos os argumentos

- Só que cada uma dessas expressões pode ter efeitos colaterais

- A avaliação dos argumentos é uma ação que produz uma *lista* de valores:

```
def evalArgs(funs: Env[Fun1], paramargs: List[(String,Exp)], env: Env[Valor]):
    Acao[List[Valor]] = paramargs match {
  case (nome, exp) :: t => for {
    varg <- if (nome.charAt(0) == ' ') id(Thunk(env, exp))
            else exp.eval(funs)(env)
    vargs <- evalArgs(funs, t, env)
  } yield varg :: vargs
  case List() => id(List())
}
```

Exceções

- Vários erros podem acontecer em nossos programas: fazer aritmética com valores que não são números, chamar coisas que não são funções, ou com o número de parâmetros errados, tentar atribuir ou dereferenciar valores que não são referências...

- Todos esses erros atualmente abortam a execução, usando a primitiva *sys.error* de Scala

- Mas e se quisermos poder detectar e recuperar esses erros na própria linguagem?