Linguagens de Programação

Fabio Mascarenhas - 2017.2

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

Abstraindo a continuação

- Podemos representar uma continuação usando uma função:

```
type Cont[T] = T => Resp
```

- Onde `Resp` (de *resposta*) é parecido com o que o tipo `Acao` do nosso interpretador era. O tipo `Acao[T]` passa a ser:

```
type Acao[T] = Cont[T] => Resp
```

- Ou seja, uma ação agora recebe uma continuação, e nos dá uma resposta que (espera-se) leva essa continuação em conta

Primitivas

- A definição de `lift` para as novas ações é simples:

```
def lift[T](v: T): Acao[T] = k => k(v)
```

- Para as outras primitivas basta passar o resultado para a continuação, ao invés de retorná-lo

```
def le(r: End): Acao[Valor] = k => (sp, m) => k(m(r))(sp, m)
def escreve(r: End, v: Valor): Acao[Valor] =
        k => (sp, m) => k(v)(sp, m + (r -> v))
```

Bind

- Na definição de `bind` vemos como estamos passando o controle do sequenciamento para “dentro” da ação:

```
def bind[T, U](a: Acao[T], f: T => Acao[U]): Acao[U] = k => a(v => f(v)(k))
```

- `bind` passa para a primeira ação uma continuação em que obtém a nova ação a partir do valor e de `f` e a chama com a continuação do `bind`

- Para entender por que essa definição, vamos ver o que acontece quando fazemos:

```
bind(escreve(0, 2), _ => le(0))
```

Desenrolando *bind*

```
bind(escreve(0, 2), _ => le(0))
```

NO WORKSHEET!

try/catch e throw com continuações

- Podemos refazer as exceções usando continuações, uma maneira para isso é 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 alternativa é trycatch executar o bloco catch passando uma continuação “identidade”, e caso o resultado tenha sido não tenha sido um erro usar a continuação original