Linguagens de Programação

Fabio Mascarenhas - 2013.1

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

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

- Claro que usar sys.error para abortar é uma espécie de trapaça...

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

Erros

- Uma Acao[T] não vai produzir mais T, mas um valor *Talvez[T]*', que é como *Option[T]* com uma mensagem associada ao caso *None*:

```
trait Talvez[T]
case class Ok[T](v: T) extends Talvez[T]
case class Erro[T](v: Valor) extends Talvez[T]
```

- Um valor Erro[T] faz *bind* entrar em curto circuito, e não continuar com a sua outra ação

- As primitivas *id* e *le* produzem valores Ok, e uma nova primitiva *erro* produz um valor Erro com alguma mensagem, usando um novo valor *StringV*

- O interpretador agora pode ser reescrito para não mais precisar de *sys.error*, e os erros continuam abortando a execução

try/catch/throw

- Uma vez no interpretador, o mecanismo de erros pode ser exposto à linguagem
- A expressão *throw(e)* produz um erro com o valor da expressão *e*
- A expressão *try e1 catch id in e2 end* executa *e1*, e caso o resultado seja um erro associa *id* ao valor do erro e então executa *e2*
- try/catch pode ser implementado em termos de uma primitiva *trycatch* que é em essência a dual de *bind*: entra em curto circuito no caso de Ok, mas passa o valor embutido no Error para *e2* em caso de erro
- Uma primitiva similar a *bind* e *trycatch* pode ser usada para implementar uma clausula *finally*

Ações sem efeitos colaterais

- Podemos retirar os efeitos colaterais de *fun*, mas manter o arcabouço das ações, com uma definição simples:

```
type Acao[T] = T
```

- Naturalmente precisamos ajustar as primitivas, e remover os recursos imperativos da linguagem, e ter de volta uma linguagem idêntica à *fun* original, só que usando *bind* como primitiva de composição, ao invés de usar os mecanismos de Scala

- Isso nos permite explorar outros modelos de como definir e compor ações; por exemplo, podemos fazer `type Acao[T] = Talvez[T]` e introduzir exceções sem outros efeitos colaterais