Linguagens de Programação

Fabio Mascarenhas - 2017.2

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...

- Em uma semântica checada, todos esses erros abortariam a execução, retornando um valor de erro

- 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 `Either[T, Erro]`, onde `Either` é como um `Option` com um valor associado ao caso `None`:

```
trait Either[A,B]
case class Left[A,B](v: A) extends Either[A,B]
case class Right[A,B](v: B) extends Either[A,B]
```

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

- As primitivas *lift* e *le* produzem valores `Left`, e uma nova primitiva `erro` produz um valor `Right(Erro(msg))` com alguma mensagem de erro

- O interpretador ainda precisa ser reescrito para checar todas as possíveis condições de erro e chamar `erro` nos locais certos

try/catch/throw - throw

- Uma vez no interpretador, o mecanismo de erros pode ser exposto à linguagem
- A expressão `throw` produz um com a mensagem passada

```
exp : ...
    | THROW STR
```

```
case class Throw(msg: String) extends Exp
```

- A avaliação de `throw` usa apenas usa a primitiva `erro` que já definimos

try/catch/throw – try/catch

- A expressão `try/catch` executa a expressão no corpo do `try`, e caso o resultado seja um erro executa a expressão no corpo do `catch`

```
exp : ...
    | TRY exp CATCH exp END
```

```
case class TryCatch(etry: Exp, ecatch: Exp) extends Exp
```

- try/catch pode ser implementado em termos de uma primitiva `trycatch` que é em essência o complemento de `bind`: entra em curto circuito no caso de etry ser `Left(v)`, mas executa ecatch se etry der `Right(e)`

try/catch/throw – finally

- Uma cláusula finally associada a um comando try garante que sua expressão sempre é avaliada independente de um erro acontecer ou não

```
exp : ...
    | TRY exp FINALLY exp END
    | TRY exp CATCH exp FINALLY exp END
```

```
case class TryFinally(etry: Exp, efin: Exp) extends Exp
case class TryCatchFinally(etry: Exp, ecatch: Exp, efin: Exp) extends Exp
```

- Primitivas similares a `bind` e `trycatch` definem como `finally` funciona

Exceções small-step

- Não é difícil adicionar tratamento de exceções ao interpretador small-step
- O termo `Throw(msg)` simplesmente avalia em um passo para `Erro(msg)`
- `TryCatch`, `TryFinally` e `TryCatchFinally` simplesmente tratam `Erro` de modo diferente das outras expressões modo: um `Erro` em `etry` faz `TryCatch` e `TryCatchFinally` reduzirem para a expressão `ecatch`, e `TryFinally` reduzir para a expressão `efin`
- Um `Erro` no bloco `catch` de `TryCatchFinally` faz ele reduzir para a expressão `efin` também