Linguagens de Programação

Fabio Mascarenhas - 2015.2

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

Erros

- Algumas operações de fun só são válidas para determinados operandos: soma, multiplicação e menor só são válidas para números, e um condicional só é válido se a condição for booleana

```
case Soma(e1, e2) => {
  val (NumV(v1), NumV(v2)) = (eval(e1), eval(e2))
  NumV(v1 + v2)
}
```

- Uma definição como a acima, que assume que os operandos estão corretos, diz que operações com operandos inválidos são *indefinidas*

- Uma operação indefinida não tem resultado

- Em geral, linguagens com operações indefinidas contam com um *verificador de tipos* que rejeita programas que poderiam ter operações indefinidas em tempo de execução

Avaliação checada

- Uma alternativa a operações indefinidas é levar erros em conta na própria definição, adicionando um *valor de erro* (e sua forma normal correspondente em uma definição small-step)

```
case Soma(e1, e2) => (evalc(e1), evalc(e2)) match {
  case (NumV(v1), NumV(v2)) => NumV(v1 + v2)
  case _ => ErroV()
}
```

- Na definição small-step:

```
case Soma(_, True()) => Erro()
case Soma(_, False()) => Erro()
case Soma(True(), _) => Erro()
case Soma(False(), _) => Erro()
```

- Sem checagem como acima, uma operação indefinida leva a avaliação small-step a ficar *stuck* (travada): uma expressão avalia em um passo para ela mesma sem ser uma forma normal

Funções de primeira ordem

- Agora podemos adicionar definições de funções à nossa linguagem
- Para começar, as definições de funções não serão expressões, e nem funções serão valores
- Um programa *fun* de primeira ordem é uma sequência de definições de funções, seguida por uma expressão
- Cada definição de função tem uma lista de parâmetros formais e o corpo da função (também uma expressão!)
- Finalmente, uma *aplicação* é uma expressão que chama uma função com uma lista de expressões (os argumentos)

Sintaxe

- Sintaxe concreta e abstrata para funções de primeira ordem:

```
prog   : {fun} exp
fun    : FUN ID '(' params ')' exp END
exp    : ...
       | ID '(' exps ')'
params : ID { ',' ID }
       |
exps   : exp { ',' exp }
       |
```

aplicação

```
case class Fun1(nome: String, params: List[String], corpo: Exp)
case class Prog(defs: Set[Fun1], corpo: Exp)
case class Var(nome: String) extends Exp
case class Ap1(fun: String, args: List[Exp]) extends Exp
```

Interpretador

- O interpretador de *fun* agora precisa do conjunto de funções que estão definidas
- Podemos interpretar uma aplicação de função usando o modelo de *substituição*
  - Primeiro avaliamos cada argumento da aplicação
  - Depois substituímos cada parâmetro pelo respectivo valor no corpo da função
  - Então avaliamos o corpo
- Small-step vs. big-step: ordem de avaliação

Aridade

- O que deve acontecer se o número de parâmetros não bate com o número de argumentos?
- O número de parâmetros de uma função é a sua *aridade*
- Geralmente, número de argumentos incompatível com a aridade é um erro (ou operação indefinida), mas outras linguagens podem se comportar de outras maneiras
  - Ignorar argumentos extras, substituir parâmetros que faltam por algum valor *default*...
  - Podem existir também funções com *aridade variável*, onde os argumentos são empacotados em uma lista

Implementando a substituição

- A substituição é uma transformação de `Exp` para `Exp`
- Para simplificar, vamos usar uma estrutura de dados de Scala que não vimos ainda: `Map[K,V]`, ou *mapeamento*
- Um `Map` é um mapeamento entre valores do tipo `K` (as *chaves*) para valores do tipo `V` (os *valores*)
- Existem duas operações básicas em Maps: busca e adição

```
scala> val m = Map(2 -> "Foo", 3 -> "Bar")
m: Map[Int,String] = Map(2 -> "Foo", 3 -> "Bar")
scala> m.get(3)
res4: Option[String] = Some("Bar")
scala> m + (5 -> "Bar")
res6:[Int,String] = Map(2 -> "Foo", 3 -> "Bar", 5 -> "Bar")
scala> m.get(5)
res7: Option[String] = None
```

O que substituir?

- A substituição vai afetar os nomes no corpo da função, mas não todos!
- Nomes que identificam funções nas aplicações não devem ser afetados
  - Espaços de nomes separados para funções e variáveis, comum em linguagens de primeira ordem
- Nomes que não têm nenhum argumento no mapa de substituição são erros!
- Não existe escopo global em *fun*
- Aliás, não existe escopo nenhum!