

Tópicos em LP

Fabio Mascarenhas – 2018.1

<http://www.dcc.ufrj.br/~fabiom/comp2>

Parâmetros de tipos

- A ideia do *polimorfismo paramétrico* é poder ter *parâmetros* nos tipos, que são “buracos” onde podemos encaixar qualquer tipo
- Representamos esses parâmetros com *parâmetros de tipos*
- Um tipo polimórfico tem um ou mais parâmetros como parte dele (podemos usar o mesmo parâmetro mais de uma vez!)

Generalização

- Criamos um tipo polimórfico por um processo de *generalização*
- Devemos tomar cuidado para verificar se os parâmetros de tipos usados foram introduzidos

```
local foo = function <a> (x: a): {number}
  local y = seq_new(x)
  return seq_new(1) .. y
end
```

```
local bar = function <a> (x: a): number
  local z = function (y: a): a
    return x
  end
  return z(1)
end
```

{ { a } }

Especialização

- O inverso da generalização é a *especialização*, que precisamos fazer toda vez que queremos *usar* um valor de um tipo polimórfico: no nosso caso, chamar uma função polimórfica, ou passar uma função com tipos polimórficos como argumento
- Na especialização, substituímos os parâmetros livres por tipos concretos
- O problema é: quais?
- Uma alternativa é incluir uma sintaxe para especialização:

```
local e = seq_new<string>()  
local s = seq_new<number>(1)  
local n = seq_byte<number>(s, 1)
```

Unificação

- Em uma chamada de função polimórfica, a ideia é usar os tipos dos argumentos para dizer quais devem ser os tipos que vamos usar na especialização
- Para fazer a atribuição de parâmetros a tipos usamos um algoritmo que mistura casamento de padrões com atribuição: a *unificação*

Unificação - Variáveis

- A unificação é um algoritmo fácil de definir quando trocamos nossos parâmetros de tipo por *variáveis de tipo*
- Uma variável inicialmente não está associada a nenhum tipo, mas pode vir a ficar durante o processo de unificação
- Uma vez associada, ela não pode mais mudar; efetivamente ela assume aquele tipo
- Duas (ou mais) variáveis não associadas podem se unir: quando uma for associada, a outra também fica associada ao mesmo tipo

Unificação - pseudocódigo

```
function unify(t1, t2):
  t1 = prune(t1)
  t2 = prune(t2)
  case t1, t2:
    match var, _:
      if occurs(t1, t2):
        error("ciclo")
      else.
        t1.type = t2
    match _, var: unify(t2, t1)
    match base, base:
      if t1.tag ~= t2.tag:
        error("incompatible")
    match func(params1, ret1), func(params2, ret2):
      if #params1 ~= #params2:
        error("arity")
      zip(unify, params1, params2)
      unify(ret1, ret2)
    match seq(elem1), seq(elem2):
      unify(elem1, elem2)
    otherwise: error("incompatible")
```

Exercício

- Implementar a função `unify` (sem o teste `occurs`)

Inferência de tipos

- A unificação consegue achar uma associação entre variáveis e tipos que torna dois tipos iguais, caso ela exista
- Caso os dois tipos não tenham nenhuma variável em aberto, a unificação faz o mesmo trabalho da igualdade
- Podemos trocar todas as comparações de tipos que fazemos em nosso verificador por unificações, e permitir que o programador omita declarações de tipos
- Uma declaração omitida é substituída por uma variável em aberto, e a unificação vai dizer qual tipo essa declaração deveria ter