Linguagens de Programação

Fabio Mascarenhas - 2015.2

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

Corotinas (Gerador)

- Uma *corotina* é como uma função que pode suspender a sua execução, retornando ao chamador mas permitindo que a execução seja retomada do ponto onde parou:

```
fun gen()
  let n = 10 in
    while 0 < n do
      yield n;
      n := n - 1
    end
  end
end
```

```
let c = coro gen in
  resume c +
  resume c +
  resume c
end
```

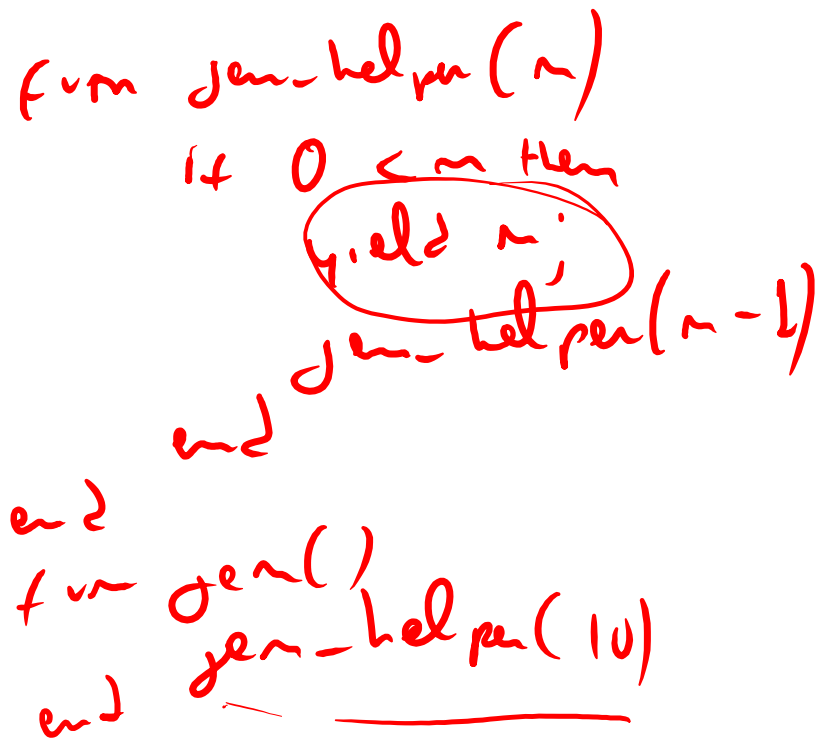
- A primitiva `coroutine` cria uma corotina a partir de uma função sem parâmetros; a primitiva `resume` inicia/retoma a execução da corotina, e a primitiva `yield` suspende a execução, passando um valor de volta para o chamador

Implementando corotinas

- Mesmo se `yield` só pode ser usado dentro do corpo da corotina não é óbvio como podemos implementar uma corotina, e é comum que yield possa ser usado por qualquer função chamada a partir da função principal da corotina

```
fun yielder(n)
  yield n
end

fun gen()
  let n = 10 in
    while 0 < n do
      yielder(n);
      n := n - 1
    end
  end
end
```



- Precisamos de alguma maneira de representar “o ponto atual da execução”

resume

- Para “saltar” para algum ponto do programa basta que guardemos a continuação daquele ponto, e então usamos ela ao invés da continuação atual
- Isso nos dá uma estratégia para implementar as corotinas e resume/yield: uma corotina é a continuação para a qual vamos saltar no resume
- Precisamos de mais um pedaço de estado global: a corotina atual
- Quando entramos em uma corotina, ela vira a corotina atual, e salvamos a anterior na própria estrutura de dados da corotina
- Corotinas não são reentrantes! Não podemos dar resume em uma corotina que não foi suspensa por um yield

yield

- Para sair de uma corotina, restauramos a corotina anterior e saltamos para a sua continuação, depois de guardar a continuação atual

- Vamos deixar o que acontece com uma corotina que chegou ao final sem ter dado um yield indefinido, por enquanto

- Em MicroC, cada corotina também precisa de sua própria seção da memória para sua pilha, e seu próprio stack pointer!

Continuation Passing Style

- Em uma linguagem com funções de primeira classe, como *fun*, podemos expor as continuações no próprio código do programa, sem precisar mudar o interpretador
- Usamos uma transformação global chamada *continuation passing style* (CPS)
- A ideia é fazer cada expressão virar uma função que recebe sua continuação (outra função), e "retorna" seu valor chamando essa continuação
- Podemos implementar corotinas diretamente na linguagem, usando referências mutáveis