

Novas funcionalidades para Łukasiewicz

Douglas Martins¹ Gustavo Zambonin² Marcello Klingelfus³

Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
INE5426 - Construção de Compiladores

¹arquiteto de sistema

²gerente de projeto, projetista de linguagem

³testador

Primeira extensão: o paradigma funcional

- ▶ Função anônima simples – `lambda type args -> expr`
- ▶ Função de alta ordem de mapeamento – `map(function, list)`
 - ▶ aplica função em todos os elementos de um iterável
- ▶ Função de alta ordem de redução – `fold(function, list)`
 - ▶ reduz elementos de um iterável para um tipo primitivo de acordo com uma função
- ▶ Função de alta ordem de seleção – `filter(function, list)`
 - ▶ remove elementos de um iterável que não respeitam uma função booleana
- ▶ Operadores de tamanho e inserção em arranjos
- ▶ Extensão da cobertura de erros semânticos
- ▶ Exemplos em `test/valid/functional/*.in`

Utilização de funções de alta ordem: fold

```
int ref t[10], output[10]
output = map(lambda int ref x -> x, t)

int ref array: t (size: 10)
int ref var: output
int ref fun: t_fold (params: int ref array t)
    int ref fun: lambda (params: int ref x, int ref y)
        ret + x y
    int ref var: t_tv
    = t_tv [index] t 0
    int var: t_ti
    for: = t_ti 1, < t_ti [len] t, = t_ti + t_ti 1
    do:
        = t_tv + t_tv lambda[2 params] t_tv [index] t t_ti
    ret t_tv
= output t_fold[1 params] t
```

Utilização de funções de alta ordem: filter

```
int t[10], output[10]
output = filter(lambda int x -> x > 10, t)

int array: t (size: 10), output (size: 10)
int array fun: t_filter (params: int array t)
    bool fun: lambda (params: int x)
        ret > x 10
    int var: t_ti
    int array: t_ta (size: 0)
    for: = t_ti 0, < t_ti [len] t, = t_ti + t_ti 1
    do:
        if: lambda[1 params] [index] t t_ti
        then:
            [append] t_ta [index] t t_ti
    ret t_ta
= output t_filter[1 params] t
```

Segunda extensão: os tipos char e char[]

- ▶ char tem aspas simples, char[] tem aspas duplas
- ▶ char + char, char + char[] produz coerção
- ▶ Truncamento de palavras maiores que o tamanho do arranjo declarado
- ▶ Adiciona também fácil suporte a comentários de uma linha com #
- ▶ Exemplos em test/valid/strings/*.in

char let = 'a'	char var: let = 'a'
char wd[4], result[10]	char array: wd (size: 4), ...
wd = "luka"	= wd "luka"
result = wd + let	= result + wd [word] let
let = result[1]	= let [index] result 1
# end of input	

Transpiling para Python

- Exemplo de tradução entre linguagens:

<code>int fun math() {</code>		<code>def math():</code>
<code> int z = 0, y = 2</code>		<code> z = 0</code>
<code> x = y * 10 + x - 15</code>		<code> y = 2</code>
<code> z = x</code>		<code> x = (((y * 10) + x) - 15)</code>
<code> ret x</code>		<code> z = x</code>
<code>}</code>		<code> return x</code>

- Outros exemplos em `test/valid/**/*py`