Novas funcionalidades para Łukasiewicz

Douglas Martins[1] Gustavo Zambonin[2] Marcello Klingelfus[3]

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

[1] arquiteto de sistema
[2] gerente de projeto, projetista de linguagem
[3] testador

Primeira extensão: o paradigma funcional

- Decomposição de um problema como um conjunto de funções
- Função anônima simples – `lambda type arg: 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 recursivamente 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

Segunda extensão: os tipos `char` e `str`

- `char` tem aspas simples, `str` tem aspas duplas
- operador de concatenação +
  - `char + str`, `char + char` produz coerção
- `str` é similar a `char array`, então suporta operação de índice
- adiciona também fácil suporte a comentários de uma linha com #

```
char letter = 'a'
str word = "luka", sum
# this is a comment discarded by the parser
sum = letter + word(1)

char var: letter = 'a'
str var: word = "luka", sum
= sum + [str] letter [index] word 1
```

Integração com LLVM

- Exemplo de geração de código intermediário:

```
int fun math() {
  int z = 0, y = 2
  int x = z
  x = y * 10 + x - 15
  z = x
  ret x
}
```

```
| define i64 @math() {
| entry:
|   %z = alloca i64
|   store i64 0, i64* %z
|   %x = load i64, i64* %z
|   %tmp2 = sub i64 %x, 15
|   %tmp = add i64 20, %tmp2
|   %x1 = add i64 %tmp, %x
|   store i64 %x1, i64* %z
|   ret i64 %x1
| }
```

Transpiling para Python

- Exemplo de tradução entre linguagens:

```
int fun math() {          | def math():
  int z = 0, y = 2        |     z = 0, y = 2
  int x = z               |     x = z
  x = y * 10 + x - 15     |     x = y * 10 + x - 15
  z = x                   |     z = x
  ret x                   |     return x
}                         |
```