Klassic言語の紹介

IALAB夏合宿

株式会社ドワンゴ
株式会社エフ・コード技術顧問
水島 宏太

私は誰？

- 2011年3月IALAB修了（博士（工学））
- 専門：構文解析
  - PEG/Packrat Parsing
- Twitter ID: @kmizu
- GitHub: kmizu
- プログラミング言語（作成）オタク
  - Onion言語（2005〜）
  - Klassic言語（2015〜）

Klassic

- https://github.com/klassic/klassic
- 自作のプログラミング言語
- 実用に使えるプログラミング言語を目指して開発中

Klassicの特徴

- 式ベース
- 第一級関数
- 静的型＆型推論
  - Hindley-Milner
  - Row Polymorphism
- スペースセンシティブリテラル
  - List, Map, Set
- Java FFI

式ベース

- 最近の多くのプログラミング言語はこれ
- 評価した結果として値を返す
  - if式
  - while式
  - 関数呼び出し
  - メソッド呼び出し
  - 各種リテラル
  - , etc.

```
def lt(x, t, e) = if(x) t() else e()
// lt: (x: Boolean, t: 'a, e: 'a): 'a
```

第一級関数

- 関数を
  - 引数として渡したり
  - 返り値として返したり

```
//stopwatch: () => Unit
val time = stopwatch( => {
  sleep(1000)
}
println(time)
```

型推論(1) - よくある誤解

- 単に

```
val f = 1 + 1 // f: Int
```

となる機能(式の型を変数に付与する）ではない

- このような推論はローカル型推論と呼ばれる
- いわゆる型推論はローカル型推論のことが多い

型推論(2) - 真実

- 双方向の推論
- たとえば、

```
// fact: Int => Int
def fact(n) = {
  if(n < 2) 1 else n * fact(n - 1)
}
```

- の型が本体での使われ方「だけ」から推論
- Klassicの型推論はHindley-Milner型推論がベース
  - ML familyが採用している型推論方式

Row Polymorphism (1)

- 関連：
  - http://gihyo.jp/news/report/01/rubykaigi2016/0001
- Duck Typeぽい型を推論
- OCamlに昔からある

```
def distance(p, e) = {
  val dx = abs(double(p.x - e.x))
  val dy = abs(double(p.y - e.y))
  sqrt(dx * dx + dy * dy) // sqrt: Double => Double
}
// inferred:
distance : (
  p: { x: Int; y: Int; ...},
  e: { x: Int; y: Int; ...}
) => Double
```

Row Polymorphism (2)

```
def using(r, f) = {
  f(r)
  r.close
}
// inferred
using: <'a, 'b, 'c>(r: { close: 'a; ... } as 'b, f: 'b => 'd) => 'c
```

Listリテラル

```
println([1, 2, 3]) // List<Int>
println([
  1
  2
  3
]) → 上記と同じ。改行もセパレータ
println([
  [1 2 3]
  [4 5 6]
  [7 8 9]
]) → 上記と同じ。スペースもセパレータ
```

Mapリテラル

```
%[x: 1, y: 1] // Map<Int, Int>
%[x: 1 y: 2]  // スペースもセパレータに
%[x: 1
  y: 2] // 改行でもOK
%[x: 1, y: "F"] // 型エラー
```

Java FFI

```
val list = new java.util.ArrayList
list->add(1)
list->add(2)
println(list)
val buffer = new java.lang.StringBuffer
buffer->append("A")->append("B")->append("C")
println(buffer)
```

- Javaのオブジェクトの型のための適切な型がない
  - 将来的にはちゃんと扱いたい
- 現段階では、特殊型Dynamicとして扱っている

その他の文法

```
/* パイプライン演算子と似ているが引数の適用の仕方が違う */
[1 2 3]->map((x) => x + 1) // map([1 2 3])((x) => x + 1)
1 #cons [] // <=> cons(1, [])
```

今後の予定

- 代数的データ型のための言語機能
- ライブラリの充実
  - Row Polymorphismを活用
- コンパイル時エラーチェックを親切に
  - 今は型エラーが出たら例外を投げている
- 0.1 リリース
  - 2017年度中がひとつの目標

気になった方へ

- https://github.com/klassic/klassic

を参照

最後に

いろいろ語り合いましょう