プログラミング入門

いらいざ (@Eliza_0x)

2017-04-20

はじめに

このスライドの目的

- 数について考える
- シンプルな操作からさまざまな演算を定義する

あなたは誰

- 高校三年間趣味でプログラミングしていました
- TwitterID: @Eliza_0x
- 大学入学祝いにふるつきくんからカラーコーンいただきました
- ΔΔ カラーコーンありがとう ΔΔ

閑話休題

数ってなんだろう

みかんがいっこ、にこ、Ｎこ..

数の本質ですか？

わたしにはわかりません
(みかんではなさそうです)

木を数え続けるだけのお仕事

ものの個数を数える、というのがわたしたちにとっての一番素朴な動機ではないでしょうか

- みかんの集合
- 木の集合
- 文字の集合

悩む数学者

フランスの数学者、アンリ・ポアンカレは「数」の定義は難しく、0、1などを厳密に定義するのは難しいと説明している。
Wikipedia - 数
https://ja.wikipedia.org/wiki/%E6%95%B0

数にもいろいろある

- 複素数
- 有理数
- 自然数
- その他多数

一番簡単な数を考えよう

- 一個、二個、三個…
- 自然数が一番素朴 (?)

自然数

- 流派があるようですが、今回は0を加えた正の整数の集合とします
- $\{x|x \in \mathbb{N}, x \geq 0\}$

自然数の定義

自然数の集合 S が次の 2 つの性質をもつと仮定する

- S は 0 を含む
- 自然数 n が S に含まれるならば、$n+1$ も S に含まれる

このとき、S はすべての自然数の集合 $\mathbb{N}$ と一致する

証明

- 帰納法で簡単に証明できる

もしかして

1. 0にあたるなにか
2. $n+1$にあたる操作

以上を定義してしまえば、自然数は表現できてしまうのではないか？

数を表現する

プログラミング言語で数を表現する

実際に成り立っていることを実験して確かめることができる

はじめに0を定義する

- 0は [] とします(空集合)
- 気になるかたは集合を多重集合と読み替えてください

N+1を定義する

- N の要素に [] を一つ追加する
- 空集合を cons する

終了

具体例を挙げてみましょう

0 は [] ですね？

1は[[]]ととなります

2は[[]，[]]となります

Nは[[]，[]，[]，[]，[]，

．．．

Haskell

- 成り立っていることを Haskell で確認してみましょう。
- Haskell - https://www.haskell.org

Haskell: 0 を定義

```
[]
```

Haskell: n+1を定義

```
succ = ([] :)
```

このように定義できます

Succは何をする関数ですか？

集合のはじめに空集合を追加します

Haskell: 動作確認

```
Haskell> []
[]
Haskell> succ []
[[]]
Haskell> (succ . succ . succ) []
[[],[],[]]
```

動いた!

自然数の定義

自然数の集合 S が次の 2 つの性質をもつと仮定する

- S は 0 を含む
- 自然数 n が S に含まれるならば、$n+1$ も S に含まれる

このとき、S はすべての自然数の集合 $\mathbb{N}$ と一致する

自然数を表現できました

ペアノの公理というそうですよ

和を表現したい

できるのですか？

いいえ、このままではできません

では、どうするのでしょうか？

$n-1$にあたる操作を定義する必要があります

Haskell: pred

```
pred = tail
```

このように定義できます

Pred は何をする関数ですか？

集合のはじめの要素を取り除きます

Haskell: 動作確認

```
Haskell> pred [[], []]
[[]]
Haskell> pred [[], [], [], [], []]
[[],[],[],[]]
Haskell> pred []
*** Exception: Haskell.tail: empty list
```

エラーが出ていますよ

今回は自然数の範囲で考えているので、これでよいのです
(0から1を引くと何になるでしょうか？)

ですが、不安ですね

$$pred(n) = \begin{cases} 0 & (n = 0) \\ n - 1 & (otherwise) \end{cases}$$

これでどうでしょうか

これはどう動作しますか？

これまでと違うのは、0-1 が 0 だということです

Haskellで定義できますか？

もちろんです

Haskell: あたらしいPred

```
pred list = case list of
    [] -> []
    otherwise  -> tail list
```

実際に動いているところを見たいのですが

```
Haskell> pred []
[]
```

安心ですね

これで和が定義できますか？

そうです、やってみましょう

和を定義する

ところで、括弧ばかりで疲れてしまいました

```
display = print . length
```

しかたありませんね

これでうまく動きますか？

```
Haskell> display [[], [], []]
3
Haskell> display []
0
Haskell> (display . succ . succ) []
2
```

もちろんです

それでは和を定義しましょう

どうやって？

$3+3$ は $(3+1)+(3-1)$ ともとれますね？

そうです

もっと！

$(3+1+1)+(3-1-1)$ ですね

もっと！！

$(3+1+1+1)+(3-1-1-1)$ ですね

何をしているか説明できますか？

$n + m$ は m が 0 になるまで $m - 1$ と $n + 1$ を繰り返したものと同じ結果になります。

和

$$add(x, y) = \begin{cases} x & (y = 0) \\ succ(add(x, pred(y))) & (otherwise) \end{cases}$$

これで和が定義できます

動きをトレースしてみましょう

$$
\begin{aligned}
& add(3,3) \\
= \ & succ(add(3,2)) \\
= \ & succ(succ(add(3,1))) \\
= \ & succ(succ(succ(add(3,0)))) \\
= \ & succ(succ(succ(3)))
\end{aligned}
$$

動きをトレースしてみましょう

$$
\begin{aligned}
&= succ(succ(succ(3))) \\
&= succ(succ(4)) \\
&= succ(5) \\
&= 6
\end{aligned}
$$

Haskell: Add

```
add x y = case y of
  [] -> x
  otherwise -> succ(add x (pred y))
```

ほとんどさきほどの数式と同じですね

0 を [] で定義していることを忘れないでください!!

Haskell: 動作確認

```
Haskell> add [] []
[]
Haskell> display(add [[],[],[]] [[],[],[]])
6
Haskell> display(add [[],[],[],[],[],[],[]] [
10
```

なにが言いたいか

- 和が $+1$ と -1 と条件分岐で表現できる
- 自分で表現した数が和で問題なく機能している
- 和は**+1** の組み合わせであるということ

もっと計算

和は+1の組み合わせである

では、積は？

和の組み合わせはありませんか？

積

$$prod(x, y) = \begin{cases} 0 & (y = 0) \\ add(x, (prod(x, pred(y)))) & (otherwise) \end{cases}$$

Haskell: 積

```
prod x y = case y of
  []    -> []
  otherwise  -> add x ( prod x (pred y))
```

Haskell: 動作確認

```
Haskell> prod [[]] [[]]
[[]]
Haskell> display(prod [[], []] [[], []])
4
Haskell> display(prod [[], [], []] [[], [], [
9
```

積が和で表現できた

差

$$sub(x, y) = \begin{cases} x & (y = 0) \\ sub(pred(x), pred(y)) & (otherwise) \end{cases}$$

Haskell: 差

```
sub x y = case y of
  [] -> x
  _  -> sub (pred x) (pred y)
```

Haskell: 動作確認

```
Haskell> sub [[]] [[]]
[]
Haskell> sub [[],[],[]] [[]]
[[],[]]
Haskell> display (sub [[],[],[],[],[]] [[],[]
3
```

-1で差が表現できた

では、商も

できません

何故ですか

$\frac{2}{3}$ は自然数の範囲に収まりません。

(私達が定義した数字は自然数の範囲でしかあつかえませんからね！)

商は特殊なんですね

- succ, pred: $\mathbb{N} \to \mathbb{N}$
- add, prod, sub: $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$
- div: $\mathbb{N} \times \mathbb{N} \to \mathbb{R}$

すると、どうしましょうか

- $0-1$を表現できるように数を拡張する
- $\frac{2}{3}$を扱えるように数を拡張する

といったことがかんがえられます、しかし今日はもういいでしょう。

演算もまだまだ拡張していくことができそうですね

積の組み合わせでべき乗を表現することもできるでしょう

そして、これらを構成しているものは+1, -1, 条件分岐のみなのです

ここからが本題

今日定義した演算と数は完結した一つの世界を作っていますね

ええ、プログラミング言語みたいでしょう？

Domain specific language: Mikan

ですから、みかんの数を数えるためのプログラミング言語 (DSL) をつくりました。

Github - Mikan https://github.com/eliza0x/Mikan

なんでつくったの

ノリ

動くの？

```
$ mikan "succ [[] [] []]"
succ succ succ succ []
$ mikan "iszero? (pred (succ [[] [] []]))"
$ mikan "if (iszero? (pred (succ [[] [] []])))
false
```

察してください

敗北の原因

- 字句解析と構文解析を一体にしたこと
- ラムダ計算をしたかった

ありがとうございました

- 空集合のあつまりとして数を表現できる
- +1, -1, 条件分岐のみで和、差、積は表現できる
  - シンプルな演算から構成的に日常的な演算を構成することができる
- 数を拡張する必要が自然に発生した
- 勢いでプログラミング言語にしたかった