プログラミング言語Egisonのコンパイラの開発

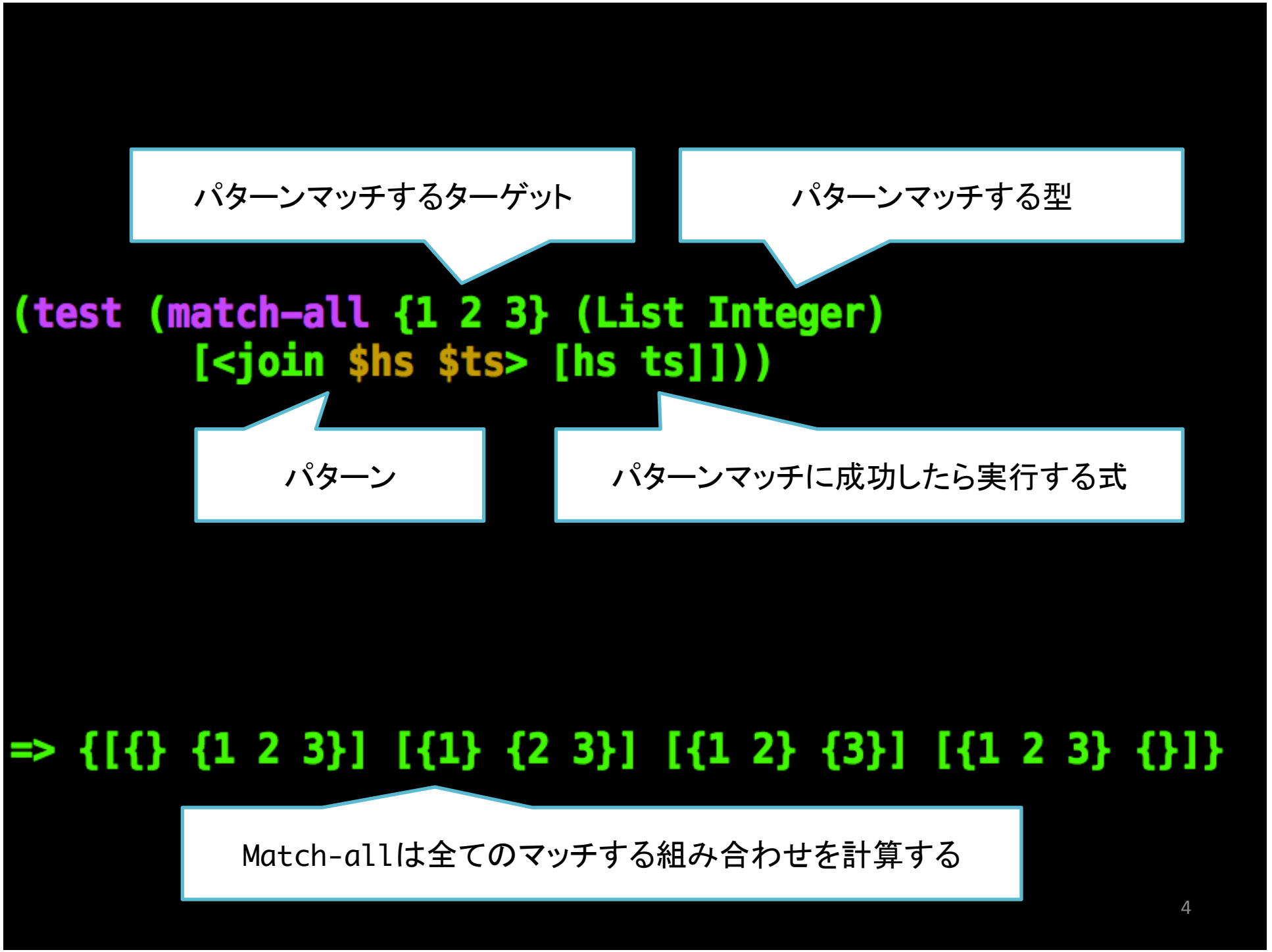
江木聡志

優れたプログラミング言語とは

- アルゴリズム記述の際の冗長性を排除
  - 人間が頭の中では当たり前に行っているのにも関わらず，プログラムでは当たり前に書けない処理があれば，そこが冗長性のある部分
- 解決されてきた冗長性
  - 関数型言語：似た関数の記述
  - オブジェクト指向：似たデータ型の定義の記述
  - ガベージコレクション：メモリ管理の記述
  - Egison：パターンマッチ（データの分解）の記述

既存の言語の問題点

- 既存のプログラミング言語では集合やマルチセットなど正規形を持たないデータのパターンマッチを自然に表現できない
  - 正規形とは同じデータに対する1つの標準的な表現のこと
  - 既存の言語では，集合やマルチセットなどをパターンマッチする際，いちいちリストに変換してパターンマッチしなければならない
- 正規形を持たないデータ型に対してのパターンマッチの一般的な方法をモジュール化する方法が必要！！


パターンマッチするターゲット
パターンマッチする型
(test (match-all {1 2 3} (List Integer)
[<join $hs $ts> [hs ts]]))
パターン
パターンマッチに成功したら実行する式
=> {[{} {1 2 3}] [{1} {2 3}] [{1 2} {3}] [{1 2 3} {}]}
Match-allは全てのマッチする組み合わせを計算する

```
(test (match-all {1 2 3} (List Integer)
        [<cons $x $ts> [x ts]]))

=> {[1 {2 3}]}
```

リストとして
パターンマッチ

```
(test (match-all {1 2 3} (Multiset Integer)
        [<cons $x $ts> [x ts]]))

=> {[1 {2 3}] [2 {1 3}] [3 {1 2}]}
```

マルチセットとして
パターンマッチ

```
(test (match-all {1 2 3} (Set Integer)
        [<cons $x $ts> [x ts]]))

=> {[1 {2 3}] [2 {1 3}] [3 {1 2}]
    [1 {2 3 1}] [2 {1 3 2}] [3 {1 2 3}]}
```

集合として
パターンマッチ

```
(test (match-all {1 2 3 4} (List Integer)
        [<join _ <cons $m <join _ <cons $n _>>>> [m n]]))
```

コレクションの2つの要素をそれぞれのパターン変数に束縛するパターン

```
=> {[1 2] [1 3] [1 4] [2 3] [2 4] [3 4]}
```

```
(test (match-all {2 8 7 2 7} (Multiset Integer)
        [<cons $m <cons ,m _>> m]))
```

2つ同じ要素を含むコレクションである場合にパターンマッチ

```
=> {2 7}
```

Egisonのパターンマッチ

- 複数の結果を持つパターンマッチ
  - 複数の形を持つ正規形を持たないデータに対するパターンマッチには必須
- パターンマッチの際に，同じパターン内で先に束縛された変数の値を参照できる（non-left-linear）
  - 意味のあるパターンを表現するのに必須
- パターンマッチの際の探索を制御できる
  - パターンマッチの際に探索において，明らかに無駄な探索をしないようにすること必要
- …他にもいろいろ

```
(define $fullhouse
  (lambda [$xs]
    (match xs (Multiset Integer)
      {[<cons $m
          <cons ,m
            <cons ,m
              <cons $n
                !<cons ,n
                   !<nil>>>>>>
          <true>]
        [_ <false>]})))
```

ここまでパターンマッチに成功したらこれ以降はバックトラックしない

ターゲットのコレクションが3つと同じ要素を持ち、
さらに残り2つの要素も同じであればパターンマッチに成功する

```haskell
fullhouse :: [Int] -> Bool
fullhouse xs = fullhouse2 $ sort xs

fullhouse2 :: [Int] -> Bool
fullhouse2 (x1:(x2:(x3:(x4:[x5]))))
  | (x1 == x2) && (x1 == x3) && (x4 == x5) = True
  | (x1 == x2) && (x3 == x4) && (x4 == x5) = True
  | otherwise = False
fullhouse2 _ = False
```

Egisonの現状

- インタプリタを実装済み
  - Hackageを使ってフリーで配布中
- マニュアルを公開中
  - 日本語でも英語でも公開中
- 論文は執筆中

使える言語Egisonを目指す

- プログラミング言語として最低限の機能は全て実装する
  - システムへのアクセス(シェルコマンド，ファイル，ネットワーク)
  - Egisonのためのパッケージシステム(面白いソフトやライブラリをユーザにも開発，共有してもらう)
- 有名になる
  - 多くのひとに使ってもらうため
- コンパイラを作る
  - 速い処理系にするため