プログラミング言語Egison

- 表現の新たな抽象化の発見 -

ソフトウエアジャパンアワード受賞スピーチ

Vol.01　Feb/3/2015

Satoshi Egi (江木 聡志)

Rakuten Institute of Technology
http://rit.rakuten.co.jp/

自己紹介

名前	江木聡志
所属	楽天技術研究所
Webサイト	http://www.egison.org/~egi/
GitHub	https://github.com/egisatoshi
研究分野	プログラミング言語理論、自動推論
趣味	Egisonプログラミング、エッセイ執筆、数学の勉強

目次

- **Egisonを作った理由**
- **Egison概要・デモ**
- **Egison開発史**
- **Egisonのこれから**

目次

- **Egisonを作った理由**
- Egison概要・デモ
- Egison開発史
- Egisonのこれから

プログラミング言語とは？

- 単純な命令列しか理解できないコンピュータに対する要求を、人間が**形式的**に記述するための言語

- e.g. C, Java, Ruby, Lisp, Haskell, Egison, …

形式的表現を考えるのは難しい

- 人間の考えをコンピュータ向けに翻訳して記述せねばならない場面は未だにある

Egisonを作った理由

- 人間の頭の中の認識 (直感) をより直接的に表現する新しい方法を発見した！

目次

- Egisonを作った理由
- **Egison概要・デモ**
- Egison開発史
- Egisonのこれから

プログラミング言語Egison

Egisonはオープンソースのプログラミング言語

https://github.com/egison/egison

パラダイム	**パターンマッチ指向**、純粋関数型
作者	江木聡志
ライセンス	MIT
バージョン	3.5.6 (2015/2/3)
最初のリリース	2011/5/24
拡張子	.egi
実装言語	Haskell 3,800行

Egisonとは？

Egisonは1つの定まった標準形を持たないデータに対しても柔軟なパターンマッチが表現可能なプログラミング言語

```
pairs = []
(1..xs.length).each do |i|
  (i+1..xs.length).each do
|j|
    if xs[i] == xs[j]
      pairs.push(xs[i])
    end
  end
end
```

Ruby

コレクション‘xs’の要素で2回以上現れる要素を列挙するプログラム

多重集合(multiset) : 要素の重複は考慮するが、順番を考慮しないコレクション型

```
(match-all xs (multiset integer)
  [<cons $x <cons ,x _>> x])
```

Egison

Egisonデモ

- ポーカーの役判定
- 麻雀の上がり判定
- 双子素数の列挙
- 巡回セールスマン問題

デモ：ポーカーの役判定

```
(define $poker-hands
  (lambda [$cs]
    (match cs (multiset card)
      {[<cons <card $s $n>
        <cons <card ,s ,(- n 1)>
         <cons <card ,s ,(- n 2)>
          <cons <card ,s ,(- n 3)>
           <cons <card ,s ,(- n 4)>
            <nil>>>>>>
        <Straight-Flush>]
       [<cons <card _ $n>
        <cons <card _ ,n>
         <cons <card _ ,n>
          <cons <card _ ,n>
           <cons <card _ ,n>
            <nil>>>>>>
        <Four-of-Kind>]
       [<cons <card _ $m>
        <cons <card _ ,m>
         <cons <card _ ,m>
          <cons <card _ $n>
           <cons <card _ ,n>
            <nil>>>>>>
        <Full-House>]
       [<cons <card $s _>
        <cons <card ,s _>
         <cons <card ,s _>
          <cons <card ,s _>
           <cons <card ,s _>
            <nil>>>>>>
        <Flush>]
       [<cons <card _ $n>
        <cons <card _ ,(- n 1)>
         <cons <card _ ,(- n 2)>
          <cons <card _ ,(- n 3)>
           <cons <card _ ,(- n 4)>
            <nil>>>>>>
        <Straight>]
       [<cons <card _ $n>
        <cons <card _ ,n>
         <cons <card _ ,n>
          <cons _
           <cons _
            <nil>>>>>>
        <Three-of-Kind>]
       [<cons <card _ $m>
        <cons <card _ ,m>
         <cons <card _ $n>
          <cons <card _ ,n>
           <cons _
            <nil>>>>>>
        <Two-Pair>]
       [<cons <card _ $n>
        <cons <card _ ,n>
         <cons _
          <cons _
           <cons _
            <nil>>>>>>
        <One-Pair>]
       [<cons _
        <cons _
         <cons _
          <cons _
           <cons _
            <nil>>>>>>
        <Nothing>]}))))
```

の部分がパターン

デモ：ポーカーの役判定（Straight Flush）

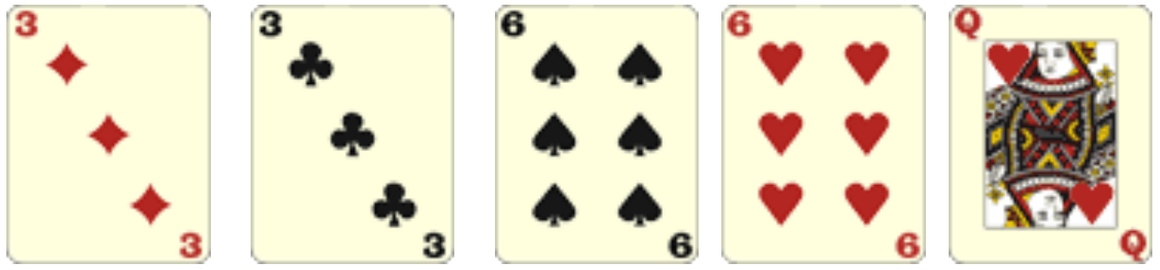
```
(define $poker-hands
  (lambda [$cs]
    (match cs (multiset card)
      {[<cons <card $s $n>
        <cons <card ,s ,(- n 1)>
         <cons <card ,s ,(- n 2)>
          <cons <card ,s ,(- n 3)>
           <cons <card ,s ,(- n 4)>
            <nil>>>>>>
       <Straight-Flush>]
       [<cons <card _ $n>
        <cons <card _ ,n>
```

デモ：ポーカーの役判定（Straight Flush）

```
(define $poker-hands
  (lambda [$cs]
    (match cs (multiset card)
      {[<cons <card $s $n>
        <cons <card ,s ,(- n 1)>
          <cons <card ,s ,(- n 2)>
            <cons <card ,s ,(- n 3)>
              <cons <card ,s ,(- n 4)>
                <nil>>>>>>>
       <Straight-Flush>]
      [<cons <card _ $n>
        <cons <card _ ,n>
```

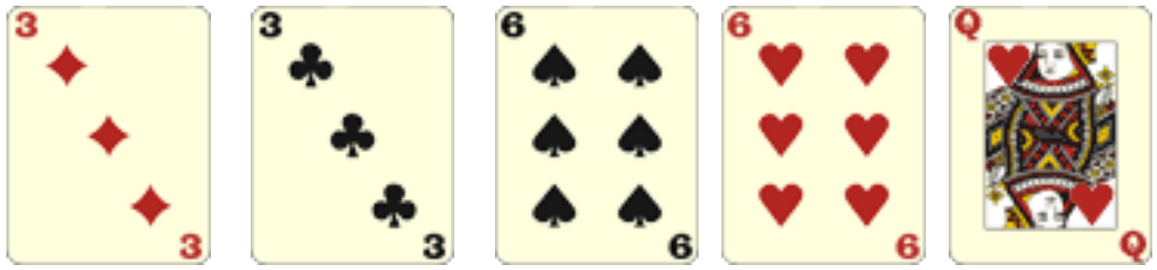
同じ名前の変数が、1つのパターン内に複数現れるようなパターンを**非線形パターン**と呼ぶ

デモ：ポーカーの役判定（Two Pair）



```
 <Three-of-Kind>]
[<cons <card _ $m>
  <cons <card _ ,m>
    <cons <card _ $n>
      <cons <card _ ,n>
        <cons
          _
          <nil>>>>>>
  <Two-Pair>]
[<cons <card _ $n>
```

デモ：ポーカーの役判定（Two Pair）



```
<Three-of-Kind>]
[<cons <card _ $m>
  <cons <card _ ,m>
    <cons <card _ $n>
      <cons <card _ ,n>
        <cons _
          <nil>>>>>>
  <Two-Pair>]
[<cons <card      $n>
```

‘_’ はどんな値にもマッチするワイルドカードと呼ばれるパターン

デモ：麻雀のあがり判定

```
;;
;; 雀頭、順子、刻子のパターンのモジュール化
;;
(define $twin
  (pattern-function [$pat1 $pat2]
    <cons (& $pat pat1)
     <cons ,pat
      pat2>>))
```

雀頭にマッチするパターン e.g. 

```
(define $shuntsu
  (pattern-function [$pat1 $pat2]
    <cons (& <num $s $n> pat1)
     <cons <num ,s ,(+ n 1)>
      <cons <num ,s ,(+ n 2)>
       pat2>>>))
```

順子にマッチするパターン e.g. 

```
(define $kohtsu
  (pattern-function [$pat1 $pat2]
    <cons (& $pat pat1)
     <cons ,pat
      <cons ,pat
       pat2>>>))
```

刻子にマッチするパターン e.g. 

```
;;
;; 上がり判定
;;
```

上がりにマッチするパターン e.g. 

```
(define $complete?
  (match-lambda (multiset tile)
    {[(twin $th_1
       (| (shuntsu $sh_1 (| (shuntsu $sh_2 (| (shuntsu $sh_3 (| (shuntsu $sh_4 <nil>)
                                                                (kohtsu $kh_1 <nil>)))
                                              (kohtsu $kh_1 (kohtsu $kh_2 <nil>))))
                                 (kohtsu $kh_1 (kohtsu $kh_2 (kohtsu $kh_3 <nil>)))))
          (kohtsu $kh_1 (kohtsu $kh_2 (kohtsu $kh_3 (kohtsu $kh_4 <nil>)))))
       (twin $th_2 (twin $th_3 (twin $th_4 (twin $th_5 (twin $th_6 (twin $th_7 <nil>)))))))
      #t]
     [_ #f]}))
```

デモ：麻雀のあがり判定

```
;;
;; 雀頭、順子、刻子のパターンのモジュール化
;;
(define $twin
  (pattern-function [$pat1 $pat2]
    <cons (& $pat pat1)
     <cons ,pat
      pat2>>))
```

← 雀頭にマッチするパターン e.g. 

```
(define $shuntsu
  (pattern-function [$pat1 $pat2]
    <cons (& <num $s $n> pat1)
     <cons <num ,s ,(+ n 1)>
      <cons <num ,s ,(+ n 2)>
       pat2>>>))
```

← 順子にマッチするパターン e.g. 

```
(define $kohtsu
  (pattern-function [$pat1 $pat2]
    <cons (& $pat pat1)
     <cons ,pat
      <cons ,pat
       pat2>>>))
```

← 刻子にマッチするパターン e.g. 

```
;;
;; 上がり判定
;;
```

上がりにマッチするパターン e.g. 

↓

```
(define $complete?
  (match-lambda (multiset tile)
    {[(twin $th_1
       (| (shuntsu $sh_1 (| (shuntsu $sh_2 (| (shuntsu $sh_3 (| (shuntsu $sh_4 <nil>)
                                                                (kohtsu $kh_1 <nil>)))
                                              (kohtsu $kh_1 (kohtsu $kh_2 <nil>))))
                                 (kohtsu $kh_1 (kohtsu $kh_2 (kohtsu $kh_3 <nil>)))))
          (kohtsu $kh_1 (kohtsu $kh_2 (kohtsu $kh_3 (kohtsu $kh_4 <nil>)))))
       (twin $th_2 (twin $th_3 (twin $th_4 (twin $th_5 (twin $th_6 (twin $th_7 <nil>)))))))
      #t]
     [_ #f]}))
```

パターンのモジュール化も可能

デモ：双子素数の列挙

の部分が双子素数にマッチするパターン

```
(define $twin-primes
  (match-all primes (list integer)
    [<join _ <cons $p <cons ,(+ p 2) _>>>
     [p (+ p 2)]]))
```

```
(take 5 twin-primes)
;=>{[3 5] [5 7] [11 13] [17 19] [29 31]}
```

先頭5個の双子素数を列挙

デモ：双子素数の列挙

の部分が双子素数にマッチするパターン

```
(define $twin-primes
  (match-all primes (list integer)
    [<join _ <cons $p <cons ,(+ p 2) _>>>
     [p (+ p 2)]]))
```

```
(take 5 twin-primes)
;=>{[3 5] [5 7] [11 13] [17 19] [29 31]}
```

← 先頭5個の双子素数を列挙

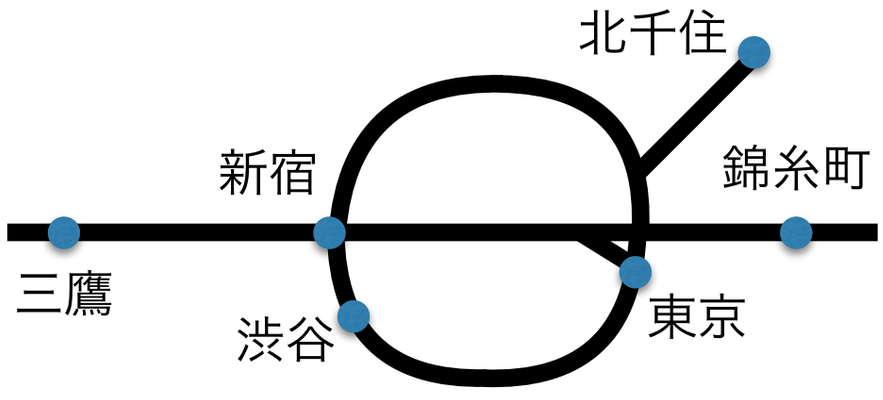
複数の結果を持つパターンマッチも可能

デモ : 巡回セールスマン問題

```
(define $station string)  ; 駅は文字列として表現
(define $price   integer) ; 運賃は整数値として表現
(define $graph   (multiset [station (multiset [station price])])) ; 運賃グラフのマッチャー

(define $graph-data ; 運賃データ
  {
   ["東京"   {                ["新宿" 200] ["渋谷" 200] ["三鷹" 390] ["錦糸町" 160] ["北千住" 220]}]
   ["新宿"   {["東京" 200]                 ["渋谷" 160] ["三鷹" 220] ["錦糸町" 220] ["北千住" 310]}]
   ["渋谷"   {["東京" 200] ["新宿" 160]                 ["三鷹" 310] ["錦糸町" 220] ["北千住" 310]}]
   ["三鷹"   {["東京" 390] ["新宿" 220] ["渋谷" 310]                  ["錦糸町" 470] ["北千住" 550]}]
   ["錦糸町" {["東京" 160] ["新宿" 220] ["渋谷" 220] ["三鷹" 470]                   ["北千住" 220]}]
   ["北千住" {["東京" 220] ["新宿" 310] ["渋谷" 310] ["三鷹" 550] ["錦糸町" 220]                  }]
   })

(define $trips ; 全ての駅を巡回して"東京"駅に戻ってくる経路をパターンマッチを用いて全て列挙
  (match-all graph-data graph
    [<cons [,"東京" <cons [$s_1 $p_1] _>]
           (loop $i [2 5]
             <cons [,s_(- i 1) <cons [$s_i $p_i] _>]
                   ...>
             <cons [,s_5 <cons [(& ,"東京" $s_6) $p_6] _>]
                   _>)>
     [(sum (map (lambda [$i] p_i) (between 1 6)))
      s]]))
```



の部分が全ての駅を巡回する経路にマッチするパターン

```
(define $main
  (lambda [$args]
    (do {[(print "経路一覧:")]
         [(each (compose show print) trips)]                              ; 全ての経路を出力
         [(write "最安値:")]
         [(print (show (min (map (lambda [$x $y] x) trips))))]})))  ; 最安値を出力
```

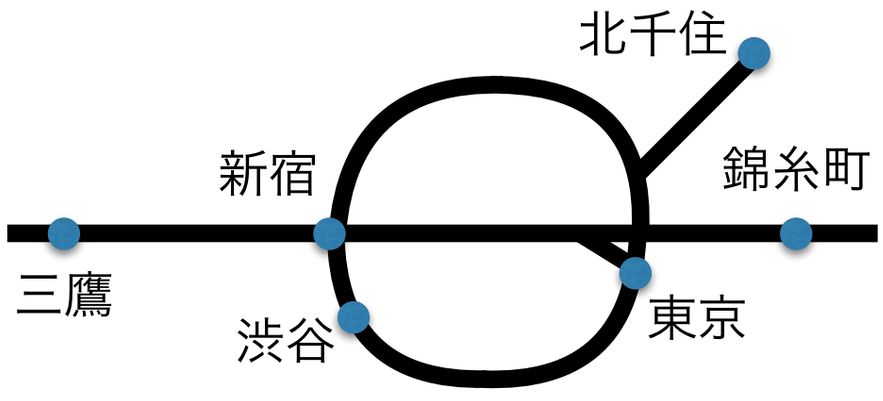
デモ：巡回セールスマン問題

```
(define $station string)  ; 駅は文字列として表現
(define $price   integer) ; 運賃は整数値として表現
(define $graph   (multiset [station (multiset [station price])])) ; 運賃グラフのマッチャー

(define $graph-data ; 運賃データ
  {
   ["東京"   {                ["新宿" 200] ["渋谷" 200] ["三鷹" 390] ["錦糸町" 160] ["北千住" 220]}]
   ["新宿"   {["東京" 200]                 ["渋谷" 160] ["三鷹" 220] ["錦糸町" 220] ["北千住" 310]}]
   ["渋谷"   {["東京" 200] ["新宿" 160]                 ["三鷹" 310] ["錦糸町" 220] ["北千住" 310]}]
   ["三鷹"   {["東京" 390] ["新宿" 220] ["渋谷" 310]                  ["錦糸町" 470] ["北千住" 550]}]
   ["錦糸町" {["東京" 160] ["新宿" 220] ["渋谷" 220] ["三鷹" 470]                   ["北千住" 220]}]
   ["北千住" {["東京" 220] ["新宿" 310] ["渋谷" 310] ["三鷹" 550] ["錦糸町" 220]                  }]
   })

(define $trips ; 全ての駅を巡回して"東京"駅に戻ってくる経路をパターンマッチを用いて全て列挙
  (match-all graph-data graph
    [<cons [,"東京" <cons [$s_1 $p_1] _>]
           (loop $i [2 5]
             <cons [,s_(- i 1) <cons [$s_i $p_i] _>]
                   ...>
             <cons [,s_5 <cons [(& ,"東京" $s_6) $p_6] _>]
                   _>)>
     [(sum (map (lambda [$i] p_i) (between 1 6)))
      s]]))

(define $main
  (lambda [$args]
    (do {[(print "経路一覧
          [(each (compose
          [(write "最安値:"
          [(print (show (m
```



の部分が全ての駅を巡回する経路にマッチするパターン

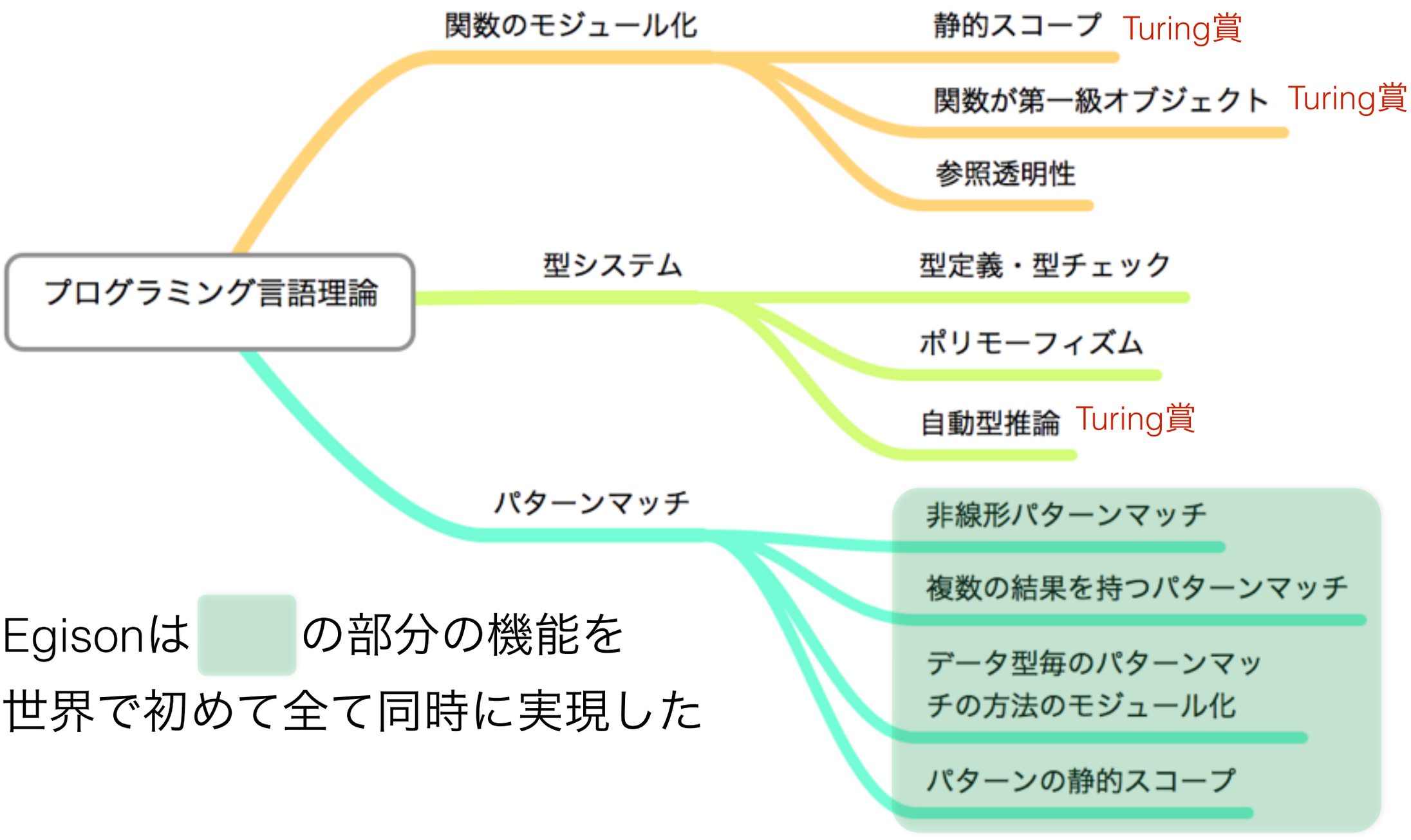
Egisonプログラマは幅広いデータ型に
対してパターンマッチの方法を定義可能

Egisonにより可能になること

- **非線形パターンマッチ**
  - e.g. ポーカーの役判定をはじめ全てのデモ
- **複数の結果を持つパターンマッチ**
  - e.g. 双子素数の列挙、巡回経路の列挙
- **データ型毎のパターンマッチの方法のモジュール化**
  - e.g. list、multisetのパターンマッチの方法の定義
- **パターンの静的スコープ**
  - e.g. 麻雀の上がり判定

Egisonの革新性

Egisonのような本質的かつ根幹的なプログラミング言語の進歩は、歴史を紐解いても数少ない



Egisonは　　の部分の機能を
世界で初めて全て同時に実現した

目次

- Egisonを作った理由
- Egison概要・デモ
- **Egison開発史**
- Egisonのこれから

発端

- 2010年3月、卒業研究のために数学の定理を自動で予想するプログラムを書いている時にEgisonのアイデアを得た

- 数学の理論の自動生成のためには、より直接的な直感を表現の方法が必要だという認識を得て、Egisonの設計を始めた

構想・最初のリリース

修士1年の期間すべてを使って構想を練り、2011年5月24日、Egison version 0.1をリリース

```
Egison Version 0.1 (C) 2011 Satoshi Egi
Welcome to Egison Interpreter!
> (match-all {1 2 3 4 5} (multiset integer) [<cons $x $ts> [x ts]])
{[1 {2 3 4 5}] [2 {1 3 4 5}] [3 {1 2 4 5}] [4 {1 2 3 5}] [5 {1 2 3 4}]}
>
Leaving Egison Interpreter.
Byebye!(^^)/
```

コミュニティの誕生

- 最初のリリースの一ヶ月後には、研究室の先輩、後輩4人が理解し、使い始めてくれた

- 新しい理論の普及のためには、周囲の優しさは非常に重要

大学院卒業

- 2012年3月、東京大学大学院を卒業（コンピュータ科学修士）

- 2012年2月-8月 IPA未踏IT人材発掘・育成事業のプロジェクトの1つとしてEgisonの開発を進める

初めての就職とEgisonチーム

- 2012年10月、初めての就職
- 2012年12月、東京大学、筑波大学のコンピュータサイエンスの学科の学生をアルバイトとして集め、Egisonチームを発足
- 2013年3月、チームでEgisonをフルスクラッチから再設計・再開発して新たなバージョンのリリース
- 2013年6月、退職、Egisonチーム解散

楽天技術研究所に入所

- 2013年11月15日、4ヶ月以上の就職活動期間を経て、楽天技術研究所に入所

- まつもとさんの紹介、萩谷先生の推薦文、竹内郁雄先生のダメ押し推薦メール

- 地に足がついた生活が送れるようになった

楽天技術研究所、入所直後のWebサイト（2013/11/15）

Japanese

Egison - Programming Language

Contents

What is Egison?

Egison is the

- pattern-matching-oriented
- purely functional

programming language.

A feature of Egison is the strong pattern matching facility. With Egison, you can represent pattern matching for

リニューアルした現在のWebサイト（2015/2/3）

Egison Documentations Libraries Installation Try It Out Blog Community Tweet 144 Star 270

The Egison Programming Language

- Express Intuition Directly with Essentially New Syntax -

Egison is a programming language that realizes non-linear pattern-matching against unfree data types.
We can directly represent pattern-matching against a wide range of data types such as lists, multisets, sets, trees and graphs.
Egison makes programming dramatically simple!

Information Processing Society of Japan
Software Japan Award 2015
Winner

```
;; Extract all twin primes from the infinite list of prime numbers with pattern-matching!
(define $twin-primes
  (match-all primes (list integer)
    [<join _ <cons $p <cons ,(+ p 2) _>>>
     [p (+ p 2)]]))

;; Enumerate first 10 twin primes
(take 10 twin-primes)
;=>{[3 5] [5 7] [11 13] [17 19] [29 31] [41 43] [59 61] [71 73] [101 103] [107 109]}
```

Pattern-Matching-Oriented

Egison proposes a new paradigm **pattern-matching-oriented**. The combination of **all of the following features** enables intuitive powerful pattern-matching.

- Non-linear patterns
- Pattern-matching with multiple results
- Modularization of the way of pattern-matching
- Pattern-matching with lexical scoping

Concept of Egison (5 min)

Online Demonstrations

- Poker Hands
- Mahjong
- Prime Numbers
- Trees
- Graphs
- Randomized 3-SAT
- Time-Series Data

View Demonstrations »

FAQ

- What is Egison?
- Why the name is Egison?
- Why new language?
- How to contribute?
- What is a difference with logic programming?
- What is the advantage over existing work?
- How to write scripts in Egison?

View Answers »

We are refining Egison paper to propagate the theory of Egison all over the world! Please check the latest copy of the paper on arXiv.org.

Egison Cheat Sheet

Please refer this cheat sheet when you write Egison programs.

Documentations

- Egison User's Manual
  - Basics of Syntax and Semantics
  - Basics of Patterns : Syntax and Semantics
- Egison Developer's Manual
  - Pattern Matching Mechanism
- Egison Paper

日本語版Webサイト（2015/2/3）

Egison　ドキュメント　ライブラリ　インストール　デモ　ブログ　コミュニティ　Tweet 49　Star 270

プログラミング言語 Egison

- 直感をそのまま表現するパターンマッチング -

情報処理学会 ソフトウエア ジャパンアワード 2015 受賞

Egison は1つの定まった標準形を持たないデータに対しても柔軟なパターンマッチが表現可能なプログラミング言語です。
リストや多重集合、集合、ツリー、グラフなどといった幅広いデータ型に対して、パターンマッチが記述できます。
それにより、Egison プログラマは非常にシンプルにプログラムを記述できるようになります。

```
;; 素数の無限リストから全ての双子素数をパターンマッチにより抽出
(define $twin-primes
  (match-all primes (list integer)
    [<join _ <cons $p <cons ,(+ p 2) _>>>
     [p (+ p 2)]]))

;; 最初の10個の双子素数を列挙
(take 10 twin-primes)
;=>{[3 5] [5 7] [11 13] [17 19] [29 31] [41 43] [59 61] [71 73] [101 103] [107 109]}
```

パターンマッチ指向

Egison は**パターンマッチ指向**という新たなパラダイムを提唱しています。**以下の3つの機能**を同時に実現することにより、強力で直感的なパターンマッチを可能にしています。

- パターンマッチの方法のモジュール化
- 複数の結果を持つパターンマッチング
- 静的スコープを保つ非線形パターン

Egison のコンセプト (5分)

デモンストレーション

多様なパターンマッチを体験してみてください。

- ポーカーの役判定
- 麻雀の上がり判定
- 素数の無限列
- ツリー
- グラフ
- 3-SATを解く乱択アルゴリズム
- 時系列データ解析

オンライン・デモを試す »

FAQ

- Egison とはどういうもの？
- Egison の名前の由来は？
- どうして新しい言語を作ったの？
- どうすれば貢献できる？
- 論理型言語との違いは？
- 他の関連研究との違いは？
- Egison でスクリプトはどう書くの？

FAQ の答えを見る »

Egisonの作者の江木が、「Egisonの設計・開発」において情報処理学会よりソフトウエアジャパンアワードを受賞しました！

[受賞理由] "従来の言語では表現不可能だった柔軟なパターンマッチを可能にする構文を生み出し、アルゴリズムをより直感的に記述できる全く新しいプログラミング言語の理論を開拓した。その理論をもとに設計・開発を進めているプログラミング言語Egisonを通して、産業界を含む日本発の世界的なコミュニティを開拓している。"(ソフトウエアジャパンアワードのWebページより引用)

Webサイト更新の結果

- 2014年3月、reddit上で話題になる
  - http://www.reddit.com/domain/egison.org/
- 2014年4月、InfoWorldで取り上げられる
  - http://www.infoworld.com/article/2606823/application-development/146094-Ten-useful-programming-languages-you-might-not-know-about.html
- 2014年6月、7月、Hacker News上で立て続けに話題になる
  - https://hn.algolia.com/?q=egison#!/story/forever/prefix/0/egison%20-edison%20-elison%20-egibson

萩谷先生による東京大学でEgisonの授業の約束

目次

- Egisonを作った理由
- Egison概要・デモ
- Egison開発史
- **Egisonのこれから**

Egisonのこれから

- コンピュータサイエンスの各分野（e.g. 自動推論、プログラムの自動生成）でEgisonの理論によりブレークスルーが起こる

- Egisonのパターンマッチは、多くのプログラミング言語に当たり前に組み込まれるようになる

- 学校で全ての子供がEgisonでプログラミングを学ぶようになる

プログラミング言語の未来

- 現在は「習得が容易なプログラミング言語」が「直感をより直接的に表現できるプログラミング言語」よりも広く使われている

- 将来、コンピュータサイエンスの素養が当たり前になった世界では後者が当たり前になる

- Egisonを教育に真っ先に取り入れれば、国の将来にとって大きなアドバンテージになる

Egisonの世界的普及は時間の問題

- 理論も実装も既にできている
- ただ、いつ普及するのかはわからない
- 普及のためには多くの方々に勉強し理解して頂くことが必要
- **Egisonを多くの方々に理解して頂き、Egisonの先にあるもっと“とんでもない”研究に没頭したい**

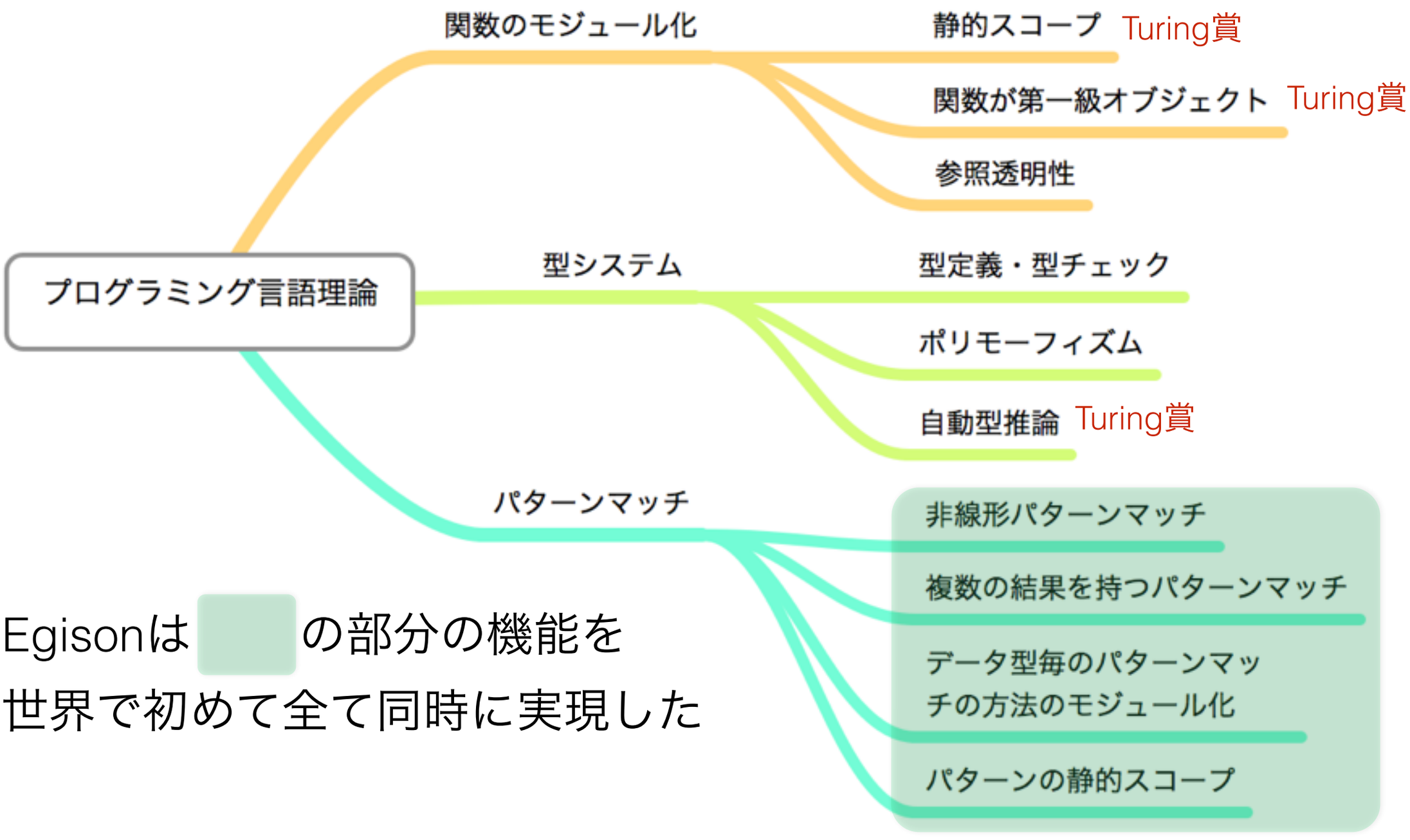
まとめ

- 「直感を直接的に表現できるプログラミング言語」こそが究極のプログラミング言語

- Egisonは最先端の「直感を直接的に表現できるプログラミング言語」

- Egisonのような本質的かつ根幹的なプログラミング言語の進歩は、歴史を紐解いても数少ない

Egisonは歴史的革新



Egisonは　　の部分の機能を
世界で初めて全て同時に実現した

関連リンク

- 公式Webサイト : http://www.egison.org/ja/
- GitHub : https://github.com/egison/egison
- Twitter : @Egison_Lang
- CodeIQ MAGAZINE連載 : https://codeiq.jp/magazine/tag/egison/
- メーリングリスト : http://www.egison.org/ja/community.html
- 個人サイト : http://www.egison.org/~egi/