

プログラミング言語Egisonの コンパイラの開発

クリエイター: 江木聡志

<http://hagi.is.s.u-tokyo.ac.jp/~egi/>

PM: 原田康徳

Egisonとは

- アルゴリズムの直感的表現を目標に設計されたプログラミング言語
 - 関数型言語
 - 遅延評価
 - 強力なパターンマッチ機能が特徴
 - 集合などのような同じデータに複数の表現型があるデータのパターンマッチも自然に表現することができる
 - 「10年後には当たり前になるように目指して欲しい」(原田PM)
 - Haskellで実装されていてHackageのパッケージとして配布されている
 - Version 0.1がリリースされてあのは2011年5月

例1.

- コレクションの要素のうち, 2つ以上含まれるものを列挙したい

例1.

- コレクションの要素のうち, 2つ以上含まれるものを列挙したい

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

例1.

- コレクションの要素のうち, 2つ以上含まれるものを列挙したい

Match-allはすべての可能なマッチについて
マッチ節の式を評価した結果のコレクションを返す

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

例1.

- コレクションの要素のうち, 2つ以上含まれるものを列挙したい

パターンマッチのターゲット

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

例1.

- コレクションの要素のうち, 2つ以上含まれるものを列挙したい

パターンマッチのターゲット

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

例1.

- コレクションの要素のうち, 2つ以上含まれるものを列挙したい

パターンマッチする型

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


例1.

- コレクションの要素のうち, 2つ以上含まれるものを列挙したい

パターンマッチする型

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

例1.

- コレクションの要素のうち, 2つ以上含まれるものを列挙したい

パターンマッチする型

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

例1.

- コレクションの要素のうち, 2つ以上含まれるものを列挙したい

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

```
-> {2 7}
```



パターン

例1.

- コレクションの要素のうち, 2つ以上含まれるものを列挙したい

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

```
-> {2 7}
```

要素のひとつとマッチするパターン変数

例1.

- コレクションの要素のうち, 2つ以上含まれるものを列挙したい

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

```
-> {2 7}
```

左でマッチした値と同じ値だったら
マッチ成功するパターン

例2.

- コレクションの要素のうち, 3つ以上連番で並んでいる要素を列挙したい

例2.

- コレクションの要素のうち, 3つ以上連番で並んでいる要素を列挙したい

```
(match-all {3 4 2 1 5 6} (Multiset Integer)
  [<cons $x
    <cons ,(- x 1)
    <cons ,(- x 2)
    <cons ,(- x 3)
    _>>>>
  x])
-> {4 5 6}
```

例2.

- コレクションの要素のうち、3つ以上連番で並んでいる要素を列挙したい

```
(match-all {3 4 2 1 5 6} (Multiset Integer)
  [<cons $x
    <cons ,(- x 1)
    <cons ,(- x 2)
    <cons ,(- x 3)
    _>>>>
  x])
-> {4 5 6}
```

パターンの左側のパターン変数に束縛された値を使って得た値ともパターンマッチできる

この半年の成果

- 言語仕様が大幅に洗練された
- コンパイラを作った
- ライブラリ・マニュアルの整備
- Workshopを開いた

この半年の成果

- 言語仕様が大幅に洗練された
- コンパイラを作った
- ライブラリ・マニュアルの整備
- Workshopを開いた

この半年間の言語仕様の改善

- Not patternの追加
- Loop patternの追加
- Macro patternの追加
- データ型定義の簡略化
- 多次元配列のパターンマッチの実現

この半年間の言語仕様の改善

- Not patternの追加
- Loop patternの追加
- Macro patternの追加
- データ型定義の簡略化
- 多次元配列のパターンマッチの実現

Not patternの追加

- ^pat : patにmatchしないときにmatch
 - E.g.

```
(match-all {1 2 1 4 2 1} (Multiset Integer)
  [<cons $x ^<cons ,x _>> x])
-> {4}
```

- 2度現れない要素がある場合にマッチ

```

(define $four-queen
  (match-all {1 2 3 4} (Multiset Integer)
    [<cons $a_1
      <cons (& ^, (- a_1 1) ^, (+ a_1 1)
        $a_2)
      <cons (& ^, (- a_1 2) ^, (+ a_1 2)
        ^, (- a_2 1) ^, (+ a_2 1)
        $a_3)
      <cons (& ^, (- a_1 3) ^, (+ a_1 3)
        ^, (- a_2 2) ^, (+ a_2 2)
        ^, (- a_3 1) ^, (+ a_3 1)
        $a_4)
      <nil>>>>
    [a_1 a_2 a_3 a_4]]))

-> {[2 4 1 3] [3 1 4 2]}

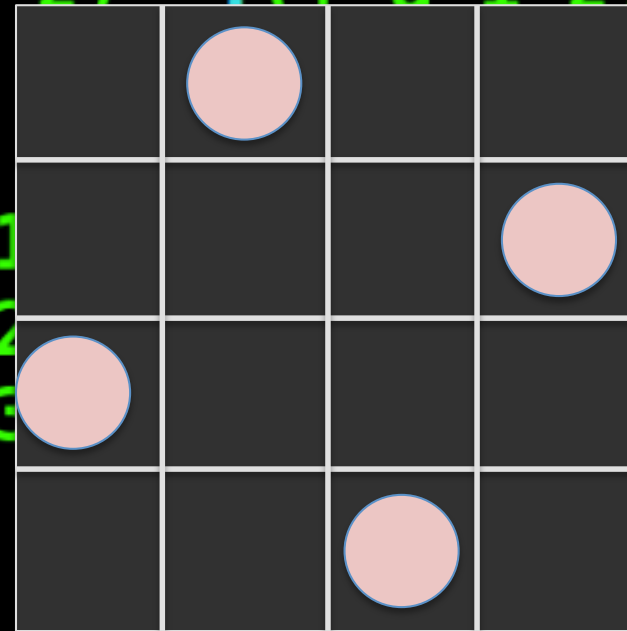
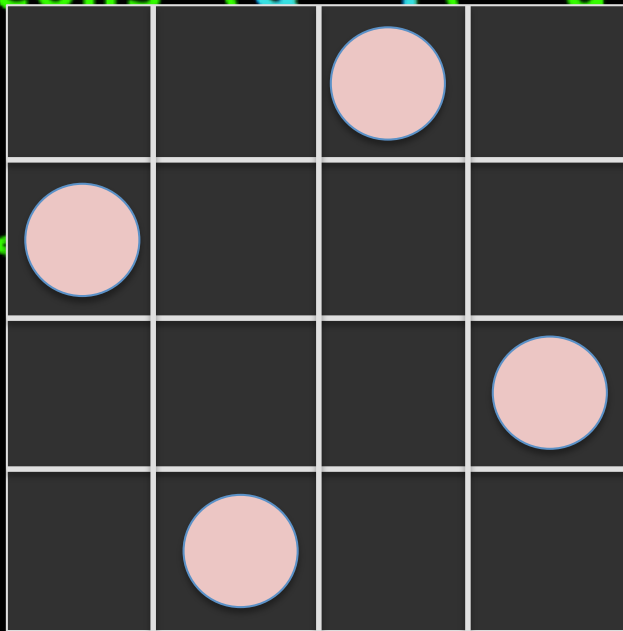
```

```

(define $four-queen
  (match-all {1 2 3 4} (Multiset Integer)
    [<cons $a_1
      <cons (& ^, (- a_1 1) ^, (+ a_1 1)
        $a_2)
      <cons (& ^, (- a_1 2) ^, (+ a_1 2)
        $a_3)
      <cons (& ^, (- a_1 3) ^, (+ a_1 3)
        $a_4)
      [a_1 a_2 a_3 a_4]]))

-> {[2 4 1 3] [3 1 4 2]}

```

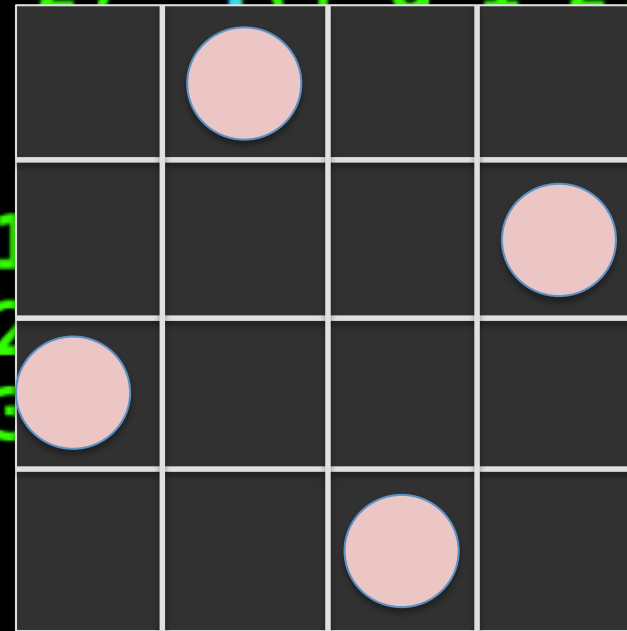
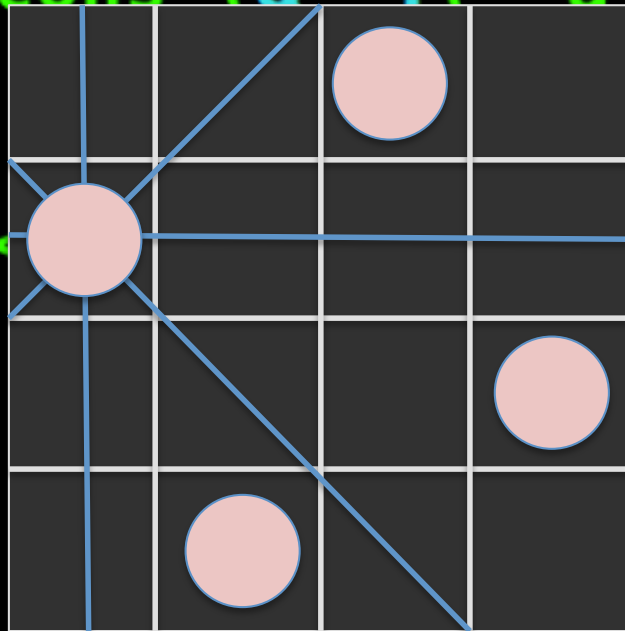


```

(define $four-queen
  (match-all {1 2 3 4} (Multiset Integer)
    [<cons $a_1
      <cons (& ^, (- a_1 1) ^, (+ a_1 1)
        $a_2)
      <cons (& ^, (- a_1 2) ^, (+ a_1 2)
        $a_3)
      <cons (& ^, (- a_1 3) ^, (+ a_1 3)
        $a_4)
      [a_1 a_2 a_3 a_4]]))

-> {[2 4 1 3] [3 1 4 2]}

```




```

(define $four-queen
  (match-all {1 2 3 4} (Multiset Integer)
    [<cons $a_1
      <cons (& ^, (- a_1 1) ^, (+ a_1 1)
        $a_2)
      <cons (& ^, (- a_1 2) ^, (+ a_1 2)
        ^, (- a_2 1) ^, (+ a_2 1)
        $a_3)
      <cons (& ^, (- a_1 3) ^, (+ a_1 3)
        ^, (- a_2 2) ^, (+ a_2 2)
        ^, (- a_3 1) ^, (+ a_3 1)
        $a_4)
      <nil>>>>
    [a_1 a_2 a_3 a_4]
    -> {[2 4 1 3] [3 1 4 2]}])

```

Not patternを用いて
斜め関係の位置にqueenがないことを表現

この半年間の言語仕様の改善

- Not patternの追加
- Loop patternの追加
- Macro patternの追加
- データ型定義の強化
- 多次元配列のパターンマッチの実現

Loop patternの追加

- パラメータの値によって繰り返しの回数が変わるパターンを表現する機構

— E.g.

```
(loop $l $i (between 1 n) <join _ <cons $a_i l>> _)
```

->

```
<join _ <cons $a_1
```

```
<join _ <cons $a_2
```

...

```
<join _ <cons $a_n _>>...>>
```

Loop patternの追加

- パラメータの値によって繰り返しの回数が変わるパターンを表現する機構

— E.g.

```
(loop $l $i (between 1 n) <join _ <cons $a_i l>> _)
```

->

```
<join _ <cons $a_1
```

```
<join _ <cons $a_2
```

...

```
<join _ <cons $a_n _>>...>>
```

このような...を表現するためのパターン

Loop patternの追加

- パラメータの値によって繰り返しの回数が変わる

添字変数: 変数名の後に続くアンダーバー'_'の後の式は評価したら整数を返す式であり, その値と変数名がセットとなって1つの変数となる.
添字は任意個つなげることができる

— E.g

```
(loop $l $i (between 1 n) <join _ <cons $a_i l>> _)
```

->

```
<join _ <cons $a_1
```

```
<join _ <cons $a_2
```

...

```
<join _ <cons $a_n _>>...>>
```

このような...を表現するためのパターン

Loop patternの追加

```
(loop $l $i {1 2} <join _ <cons $a_i l>> _)
```

Loop patternの追加

```
(loop $l $i {1 2} <join _ <cons $a_i l>> _)
```



ループ変数

Loop patternの追加

```
(loop $l $i {1 2} <join _ <cons $a_i l>> _)
```



添字変数

Loop patternの追加

```
(loop $l $i {1 2} <join _ <cons $a_i l>> _)
```



添字が動く範囲

Loop patternの追加

```
(loop $l $i {1 2} <join _ <cons $a_ i l>> _)
```

ループ式：添字が空でない場合に
この式に展開される

Loop patternの追加

```
(loop $l $i {1 2} <join _ <cons $a_i l>> _)
```

末尾式：添字が空の場合に
この式に展開される

Loop patternの追加

```
(loop $l $i {1 2} <join _ <cons $a_i l>> _)
```

->

```
<join _ <cons $a_1 (loop $l $i {2} <join _ <cons $a_i l>> _)>>
```

Loop patternの追加

```
(loop $l $i {1 2} <join _ <cons $a_i l>> _)
```

```
->
```

```
<join _ <cons $a_1 (loop $l $i {2} <join _ <cons $a_i l>> _)>>
```

```
->
```

```
<join _ <cons $a_1 <join _ <cons $a_2 (loop $l $i {} <join _ <cons $a_i l>>  
_)>>>>
```

Loop patternの追加

```
(loop $l $i {1 2} <join _ <cons $a_i l>> _)
```

->

```
<join _ <cons $a_1 (loop $l $i {2} <join _ <cons $a_i l>> _)>>
```

->

```
<join _ <cons $a_1 <join _ <cons $a_2 (loop $l $i {} <join _ <cons $a_i l>>  
_)>>>>
```

->

```
<join _ <cons $a_1 <join _ <cons $a_2 _>>>>
```

Loop patternの追加

- パラメータの値によって繰り返しの回数が変わるパターン

1からnまでの整数を含むコレクションを返す

— E.g.

```
(loop $l $i (between 1 n) <join _ <cons $a_i l>> _)
```

->

```
<join _ <cons $a_1
```

```
<join _ <cons $a_2
```

...

```
<join _ <cons $a_n _>>...>>
```

```

(define $four-queen
  (match-all {1 2 3 4} (Multiset Integer)
    [<cons $a_1
      <cons (& ^, (- a_1 1) ^, (+ a_1 1)
        $a_2)
      <cons (& ^, (- a_1 2) ^, (+ a_1 2)
        ^, (- a_2 1) ^, (+ a_2 1)
        $a_3)
      <cons (& ^, (- a_1 3) ^, (+ a_1 3)
        ^, (- a_2 2) ^, (+ a_2 2)
        ^, (- a_3 1) ^, (+ a_3 1)
        $a_4)
      <nil>>>>
    [a_1 a_2 a_3 a_4]]))

-> {[2 4 1 3] [3 1 4 2]}

```



```

(define $four-queen
  (match-all {1 2 3 4} (Multiset Integer)
    [<cons $a_1
      (loop $l $i {2 3 4}
        <cons (loop $l1 $i1 (between 1 (- i 1))
          (& ^, (- a_i1 (- i i1))
            ^, (+ a_i1 (- i i1))
            l1)
          $a_i)
        l>
      <nil>)>
    [a_1 a_2 a_3 a_4]]))

```

Loop patternを使えば先ほどのfour-queenはこんなふうにつきりする

```

(define $n-queen
  (lambda [$n]
    (match-all (between 1 n) (Multiset Integer)
      [<cons $a_1
        (loop $l $i (between 2 n)
          <cons (loop $l1 $i1 (between 1 (- i 1))
            (& ^, (- a_i1 (- i i1))
              ^, (+ a_i1 (- i i1))
                l1)
              $a_i)
            l>
          <nil>)>
        ]@ (loop $l $i (between 1 n) {a_i @l} {}))]))

```

さらにもうすこし頑張るとn-queenに一般化できる

この半年間の言語仕様の改善

- Not patternの追加
- Loop patternの追加
- Macro patternの追加
- データ型定義の強化
- 多次元配列のパターンマッチの実現

Macro patternの追加

- 頻出するパターンを再利用するための機構
 - Egisonではパターンは第一級オブジェクト
 - マッチ節のパターンの部分にパターンを返す式を書ける

```
(define $junshi
  (macro [$s $pat]
    <cons $`s <cons ,(+ `s 1) <cons ,(+ `s 2) pat>>>))

(test (match-all {1 3 5 6 2 4} (Multiset Integer)
  [(junshi %m (junshi %n _)) [m n]]))

-> {[1 4] [4 1]}
```

この半年間の言語仕様の改善

- Not patternの追加
- Loop patternの追加
- Macro patternの追加
- データ型定義の強化
- 多次元配列のパターンマッチの実現

データ型定義の強化

- ユーザがデータ型ごとにパターンマッチの方法を記述することによって, Egisonの複雑なパターンマッチを実現されている
- そのデータ型定義の表現が簡潔になりかつ強力になった！
 - より詳細にパターンマッチの方法を定義できるようになった

この半年間の言語仕様の改善

- Not patternの追加
- Loop patternの追加
- Macro patternの追加
- データ型定義の強化
- 多次元配列のパターンマッチの実現

多次元データのパターンマッチ

- これでオセロや将棋のパターンマッチも可能に！
 - ICFPC2012のLambda Mineのシミュレータをサンプルとして書いています(倉庫番のようなゲーム)
 - sample/icfpc2012/
- 対称性を意識せずにパターンマッチを表現できる
 - オセロ盤, 碁盤 : 4方向
 - ルービックキューブ : $6 \times 4 = 24$ 方向

この半年の成果

- 言語仕様が大幅に洗練された
- コンパイラを作った
- ライブラリ・マニュアルの整備
- Workshopを開いた

未踏開始時からの速度の向上

- 2.3.10 (2012/8/3), 0.4.0 (2012/2/1)
- Fact (n=30000)
 - Compiler(2.3.10) real 1.05 user 0.80 system 0.04
 - Interpreter(0.4.0) real 2.41 user 2.16 system 0.04
- List-eq (n=30000の場合)
 - Compiler(2.3.10) real 3.74 user 3.53 system 0.20
 - Interpreter(0.4.0) real 5.55 user 5.12 system 0.42
- N-queen (n=9の場合)
 - Compiler(2.3.10) real 1.61 user 1.60 system 0.00
 - Interpreter(1.2.3) real 11.05 user 11.01 system 0.03

この半年の成果

- 言語仕様が大幅に洗練された
- コンパイラを作った
- ライブラリ・マニュアルの整備
- Workshopを開いた

ライブラリやプリミティブ関数, マニュアルの整備

- ライブラリ関数の充実化・整備・効率化
- サンプルコードの充実させた
- ファイルIOなどを実装
 - Egisonでスクリプトを書けるようになりました
 - sample/io/
 - hello.egi, cat.egi, copy.egi など
- 基本的なライブラリの自動ロード機能
- Workshopに備えてEgison教材を執筆

この半年の成果

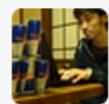
- 言語仕様が大幅に洗練された
- コンパイラを作った
- ライブラリ・マニュアルの整備
- Workshopを開いた

この半年の成果

- 言語仕様が大幅に洗練された
- コンパイラを作った
- ライブラリ・マニュアルの整備
- Workshopを開いた
 - 言語のユーザが増えた
 - 知名度が上がった
 - 公式Twitterアカウント @Egison_Langのフォロワーも約5倍に増えて57人に！
 - 開発者も増えた！

Egison Workshop

- 7月7日(土)に富士ソフトアキバプラザで行われました
 - 13:00 – 19:00 の6時間
 - 4人のEgisonistsの発表
- Haskellerたちを中心にTwitter上で話題に！



松山朋洋 @m2ym

Egisonワークショップに参加します ipa.go.jp/jinzai/mitou/2...

Expand

27 Jun



心清いマン(2歳)(ky)

@NU_MA

Follow



EgisonでQM法とか12erでできていない人は行けばいいのでは

Reply Retweet Favorited



ジャバ仙人 @plus7

1日1回Egisonの宣伝

Expand



wistery_k (2個) @wistery_k

Egisonふつうに面白かった

Expand

4 Jul

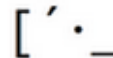


Yoichi Hirai @pirapira

ICFPCでegison勝たせればいいのだ。

Expand

2 Jul



Hideyuki Tanaka @tanakh

日本を代表するプログラミング言語になってほしい。

Expand

3 Jul



Mokele-mbembe @ynil

egisonワークショップ登録した

Expand

3 Jul

Egison Workshop


- 4人のEgisonistsによる発表
 - EgisonでQM法 本多健太郎
 - QM法は電子回路を簡略化するためのアルゴリズムです. Multisetのパターンマッチを使うためEgisonを使うと非常に簡潔に書けます.
 - Egisonで文法解析(PEG) 中村宇佑
 - Egisonの強力なパターンマッチ機能を持ちいて文法解析のパターンマッチをします. Egisonの拡張のアイデアについても話してくれました.
 - Egisonで麻雀 川又生吹
 - 麻雀の役判定もまさにMultisetのパターンマッチであるので, Egisonを使うと非常に簡潔に書けます.
 - パターンのパターンマッチング 平井洋一
 - Egisonではパターンのデータ型を定義してパターンのパターンマッチをしてEgisonで遊べます.

Egison Workshop

- Workshopが終わったあともたくさんのツイートが！

 **Hideyuki Tanaka** @tanakh 25 Jul
[' _ Egisonは日本初のとってもユニークな言語だから、がんばってほしいなあ
Expand

 **えくすわいえくす / xyx** @xyx_is 18 Jul
@__Egi egison +RTS -p で (test (loop \$! \$! (between 0 1000) {i @!} {}))
を見てみたんですが、isEmptyCollectionが167167000回で呼ばれすぎ
ている気がします
Expand Reply Retweet Favorited

 **やー** @y3eadgbe 7 Jul
うへー予想以上にEgisonすごいな
Expand

 **原田 康德** @viscuit 7 Jul
Egison ワークショップ。今までの言語仕様上の色んな謎が溶けた。
egisonの魅力にはまった一人になりました。脳が酸っぱくなるくら
い頭使ったわ。 #egison_workshop
Expand

 **xenophobia=undefined** @xenophobia_... 13 Jul
とうとう来たか！Macro実装Egison！
Expand

 **わさびず** @wasabiz 1 Aug
Egisonのワークショップの講演者のところにゼミのチューターがいて
ビビっている
Expand Reply Retweet Favorited

こんなにEgisonの魅力にはまってくれた方も

- WorkshopのためにEgisonの勉強をして使えるようになり、バグ報告までしてくれました



- Workshopが終わったあともたくさんのEgisonツイートをしています



こんなにEgisonの魅力にはまってくれた方も

- そしてなんとEgisonの機能をTemplate Haskellを使ってHaskellに組み込んだものまで作ってくれました！！



xenophobia=undefined @xenophobia__

30 Jul

とりあえず(match-all {1 2 3} (Set Integer) [<cons \$x \$xs> [x xs]])ぐらいなら走る。 [#Egison](#) [#Haskell](#)

Expand



xenophobia=undefined @xenophobia__

30 Jul

HaskellのDSLとしてegisonの式をパース・評価するQuasiQuotesを作ってみた。式クオートしか書いてないし色々未実装だけど。

[#Egison](#) [#Haskell](#)

Expand

こんなにEgisonの魅力にはまってくれた方も

- そしてなんとEgisonの機能をTemplate Haskellを使ってHaskellに組み込んだものまで作ってくれました！！

```
-- Example1
-- http://hagi.is.s.u-tokyo.ac.jp/~egi/egison/manual/n-queen.html
nqueen :: Int -> [[Int]]
nqueen = [egison| (lambda [$n]
    (match-all (between 1 n) (Multiset Integer)
      [<cons $a_1
        (loop $l $i (between 2 n)
          <cons (loop $l1 $i1 (between 1 (- i 1))
            (& ^,(- a_i1 (- i i1))
              ^,(+ a_i1 (- i i1))
                l1)
            $a_i)
          l>
          <nil>>
          {@(loop $l $i (between 1 n) {a_i @l} {})})))]
    :: Int -> [[Int]]  |]

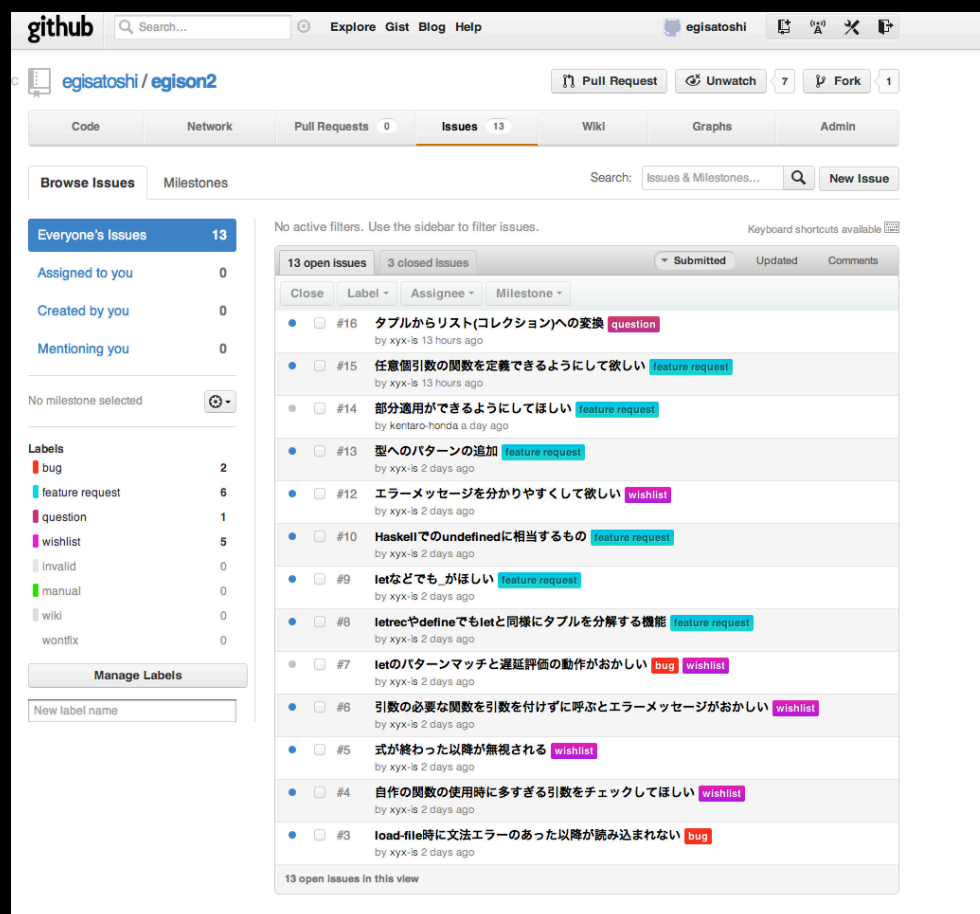
> nqueen 5
[[1,3,5,2,4],[1,4,2,5,3],[2,4,1,3,5],[2,5,3,1,4],[3,1,4,2,5],[3,5,2,4,1],[4,1,3,5,2],[4,2
```

新たな開発者たち

- 本多健太郎さん
 - 主にライブラリの最適化をしてくれています
- 中村宇佑さん
 - デバッグやプリミティブ関数の実装などをしていています

新たな開発者たち

- Githubで協力しあいながら現在開発しています



今後の予定

- Egisonをメジャーな言語にしていく
 - もっと速いコンパイラを完成させる
 - 型付きEgisonを考える
 - 本当に速いコンパイラと型はEgisonを広めるために必須だと思われる
 - 再びWorkshopを開催し言語を広める

謝辞

- 本多健太郎さん
 - 最初のEgisonistとなってくれ、深くEgisonを理解し、多くの言語仕様改善の革新的なヒント、アドバイスをくれました。またlibrary関数を効率の良いように書き直してくれました。開発者本人よりも良いEgisonコードを書きます。
- 中村宇佑さん
 - Egisonにいろいろな入力を試し、たくさんのEgisonの不具合を報告してくれました。型理論の視点からEgisonの仕様について多くのアドバイスをもらいました。
- 川又生吹さん
 - Egisonを初期から使ってくれていたEgisonistでEgisonで麻雀プログラムを始め多くのプログラムを書いてくれました。Egison開発についても多くのアドバイスをもらいました。
- 平井洋一さん
 - Egison開発について多くの重要なアドバイスをもらいました。
- 角谷良彦さん
 - EgisonのEmacs modeにコメント色付け機能を追加してくれました。
- 萩谷昌己先生
 - Egisonの新たな言語仕様改善のアイデアについて毎回アドバイスを頂きました。修士を卒業したあとも暖かく見守ってくださいました。