プログラミング言語 Egison のコンパイラの開発

−超強力なパターンマッチプログラミング−

1. 背景

Egison とは、本プロジェクトのクリエータが 2010 年 3 月にアイデアを得て開発を始め、2011 年 5 月に最初のリリースをしたプログラミング言語である。

Egison の特徴はその強力なパターンマッチの記述力にある。Egison を使うと、正規形を持たないデータ、例えば、集合や多重集合などといったコレクションや、また環や群といった代数構造などのパターンマッチを直感的に表現することができる。正規形というのは、同じデータを表現する 1 つの決まった標準的な形のことをいう。通常では、これらの正規形を持たないデータ型のデータのパターンマッチを行うには、一度これらのデータを、正規形を持つデータ型として捉え直して、パターンマッチを行うことが必要である。例えば、既存のプログラミング言語では集合のパターンマッチを行う際、これをリストとして捉え直してパターンマッチを行う。多くのプログラマは、この煩雑な作業を当たり前のことだと認識しているが、これは実はかなりのプログラミングの際の潜在的な精神的なストレスになっている。Egison では、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を用意することによって、この問題を解決した。

プロジェクト開始時の Egison はまだ実験的に作られた段階の言語で、誰でも日常的に使えるといった言語ではなかった。それを誰でも日常的に使える言語に引き上げたいというのが本プロジェクトの動機であった。

2. 目的

本プロジェクトの目的は、Egison を開発者やその知り合いに留まらずに広く一般に使える言語にまで完成度を引き上げ、Egison をより多くのユーザに普及させることを通して、パターンマッチの重要性を世に知らしめることである。

3. 開発の内容

開発した Egison の処理系は Hackage のパッケージとして配られているため、Haskell がインストールできる環境ならどこでも動かすことができる。

Egison の処理系はプロジェクト開始前からすでにあったが、プロジェクト期間中に以下の開発を行った。Egison のより詳細な仕様は参考 URL のサイトにあるマニュアルを参照していただきたい。

1. 言語仕様を改良した。具体的には以下の改良をした。
   ① Not パターン、Loop パターン、Macro パターンのような新たなパターンを追加してパターンの記述力を高めた。図 1 は N-Queen の問題を解くアルゴリズムを Egison で書いたものである。Not パターンや Loop パターンを使うと、一般的な n についてこのアルゴリズムが簡潔に表現できる。図 2 は Macro パターンを用いて麻雀の順子と刻子のパターンをモジュール化している例である。

```
(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} {})])))
```

図 1 Egison で書かれた N-Queen

```
(define $shuntsu
  (macro [$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]}

(define $kohtsu
  (macro [$s $pat]
    <cons $`s <cons ,`s <cons ,`s pat>>>))

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

-> {[1 5]}
```

図 2 Macro パターンを用いた順子、刻子のモジュール化

② Egison ではプログラマが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、型ごとの柔軟なパターンマッチを実現する。図 3 は多重集合のパターンマッチの方法を定義したものである。 Multiset は型 a を受け取って、型 a の多重集合の型を返すように定義されている。type 式の中で、パターンのパターンマッチを行なっていて、パターンごとにどのようにパターンマッチが行われるのかを記述している。プロジェクト期間中にこの記述の仕様を改良した。この改良により、記述は簡潔になったと同時により詳細なパターンマッチのアルゴリズムを記述できるようになった。また、パターンマッチの内部的な実装もシンプルになった。

2. 速度の向上のために、コンパイラを作成し、ライブラリ関数の整理などを行った。それにより、プログラムの実行速度が最高で 7 倍近く高速になった。またプリミティブ関数も整理・追加した。基本的なライブラリの自動ロード機能を追加し、使い勝手を向上させた。
3. ワークショップを開催した。Egison のマニュアル・チュートリアルを整備した。Twitter などを用いて Egison の開発状況を公開し、広報した。現在では、Egison 開発者は 4 人に増え、Egison-Quote という Egison に関連する別の独立したプロジェクトも立ち上がった。Egison-Quote を用いると Egison のプログラムを Haskell のプログラムのなかに埋め込むことができる（図 4）。

```
(define $Multiset
  (lambda [$a]
    (type
      {[,$val []
        {[$tgt (match [val tgt] [(List a) (Multiset a)]
                 {[[<nil> <nil>] {[]}]
                  [[<cons $x $xs> <cons ,x ,xs>] {[]}]
                  [[_ _] {}]})]}]
       [<nil> []
        {[{} {[]}]
         [_ {}]}]
       [<cons ,$px _> [(Multiset a)]
        {[$tgt (if ((member? a) px tgt)
                   {((remove a) tgt px)}
                   {})]}]
       [<cons _ _> [a (Multiset a)]
        {[$tgt (letrec {[$helper (lambda [$xs $ys]
                                   (match ys (List a)
                                     {[<nil> {}]
                                      [<cons $z $zs> (if ((member? a) z xs)
                                                         (helper {@xs z} zs)
                                                         {[z {@xs @zs}] @(helper {@xs z} zs)})]}))]}
                       (helper {} tgt))]}]
       [<join ,$pxs _> [(Multiset a)]
        {[$tgt (letrec {[$helper (lambda [$xs $ys]
                                   (match xs (List Something)
                                     {[<nil> ys]
                                      [<cons $z $zs> (if ((member? a) z ys)
                                                         (helper zs ((remove a) ys z))
                                                         {})]}))]}
                       {(helper pxs tgt)})]}]
       [<join _ _> [(Multiset a) (Multiset a)]
        {[$tgt
          (foldr
                 (lambda [$xi $xs]
                   (let {[[$x $i] xi]}
                     (map&concat
                                 (lambda [$sub]
                                   (do {[[$ys $zs] sub]
                                        [$zs ((remove-all a) zs x)]}
                                     (match-all (loop $l $j (between 1 i) {x @l} {}) (List a)
                                       [<join $us $vs> [{@us @ys} {@zs @vs}]])))
                                 xs)))
                 {[{} tgt]}
                 ((occurrence a) tgt))]}]
       [_ [Something]
        {[$tgt {tgt}]}]
       }))))
```

図 3 Multiset の定義を用いた多重集合のパターンマッチ

```
-- 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
```

図 4 Egison-Quote

4. 従来の技術(または機能)との相違

正規形を持たないデータ型に対する直感的なパターンマッチを実現するためには、少なくとも以下の 2 つの機能は必要である。

1. 複数のパターンマッチの結果を扱える。
2. パターンの左側で得たパターンマッチの結果をそれよりも右側のパターンマッチの処理で参照できる。

従来にも正規形を持たないデータ型のためのパターンマッチのシステムはあった。しかし上記両方の機能を満たしているパターンマッチのシステムは Egison が初めてである。

また Egison にしかない独自のパターンの概念がある。そのパターンにマッチしない場合マッチに成功する Not パターン、バックトラックを制御するための Cut パターン、パターン内の繰り返しを表現するための Loop パターンなどである。

5. 期待される効果

Egison を使うと正規形を持たないデータ型を扱うプログラムの記述が大幅に簡潔になる。集合や多重集合のような正規形を持たないデータ型は多くのプログラムに現れるので、Egison が普及すれば、Egison または Egison の機能はプログラミングの際の大きな助けになるはずである。

6. 普及(または活用)の見通し

プロジェクト期間中にワークショップを開催したりしたおかげで、Egison を知っている人、使ったことある人、使える人、使う人はかなり増えた。例えば、プロジェクト開始当初は 10 人程度しかいなかった Egison の Twitter アカウントのフォロワーが現在では 60 人を超えた。Egison はプロジェクト開始当初はクリエーター人で開発していたが、今は開発を協力してくれる方たちが現れ、小さいながらもオープンソースコミュニティができた。また Egison の機能を他の言語に組み込むためのソフトウェアを作成し公開してくれた方まで現れた。将来 Egison の備えるパターンマッチの記述力が当たり前のものとして、一般的なプログラミング言語に広まることが期待される。

7. クリエータ名(所属)

江木 聡志(東京大学大学院 情報理工学系研究科)

(参考)関連 URL

Egison 公式ホームページ: http://hagi.is.s.u-tokyo.ac.jp/~egi/egison/index.html
Hackage の Egison のページ: http://hackage.haskell.org/package/egison
Github の Egison のページ: https://github.com/egisatoshi/egison2