パターンマッチ指向プログラミング言語Egison

江木 聡志

2013年7月26日(土)

http://egison.pira.jp/files/pyfes.pdf

自己紹介

- 江木 聡志
  - サイト : http://egi.pira.jp/
  - Github : https://github.com/egisatoshi/
  - Twitter : @__Egi
  - 活動 :
    - プログラミング言語Egisonの設計・開発
      - サイト : http://egison.pira.jp/
      - ソース : https://github.com/egisatoshi/egison3
      - Twitter : @Egison_Lang
  - 受賞歴 :
    - IPAから2011年度の未踏スーパークリエーターに認定されました
  - 現在 :
    - 就職活動をしております

プレゼンテーションの構成

- プログラミング言語Egisonのこれまで
- プログラミング言語Egisonのこれから

プログラミング言語Egisonのこれから

- プログラミング言語Egison
  - Egisonとは
  - パターンマッチとは
  - Egisonのパターンマッチ
- 私のプログラミング経験

プログラミング言語Egison

- 世界で初めて集合に対するパターンマッチを直接的に表現することを可能にしたプログラミング言語
  - 「パターンマッチ指向」という新たなパラダイムを提唱した
  - オープンソース (MITライセンス)
  - Version3.0.10 (2013年7月現在)
  - 2011年5月に最初のバージョンをリリース
    - 開発自体が始まったのは2010年3月

パターンマッチとは?

- 「データの分解」とその結果に基づく「条件分岐」を簡潔に表現するためのプログラミング言語の構文

Egison以前のパターンマッチ

- 配列のようなデータの構造が「きっちり定まった」データの分解を簡潔に表現できる
  - 何重にもネストするデストラクタの適用を書かなくてよくなる
  - 例. リストのパターンマッチ

Pattern : Cons $a (Cons $b $c)
Target : Cons 1 (Cons 2 (Cons 3 Nil))

Result : $a = 1, $b = 2, $c = Cons 3 Nil

Egisonのパターンマッチ

- 集合のような「定まった」形のないデータも直接的にパターンマッチ可能
  - もし{1, 2, 3}というコレクションを集合として捉えたら、{1, 3, 2}や{3, 2, 1}も{1, 2, 3}と同じコレクションを表す
  - 何重にもネストするループがパターンマッチに置き換えられる

```
for (…) {
  for (…) {
    if (xs[i] == xs[j]) return xs[i]
    …
  }
}
```

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

Egisonを作った理由

- 人間の推論を定式化してコンピュータで動かしたい
  - 整数論や初等幾何学の公理を与えると自動で定理を予想し、自動で証明を考えていくプログラム
- 人間の認識をコンピュータ上で表現できるように定式化したい
  - 既存のプログラミング言語では、どれも集合のようなデータを直接的に扱えない
  - このような表現体系の上では、人間の直感を表現できるはずがない
- 集合のようなデータでも直感的に扱えるプログラミング言語を作った

Egisonのパターンマッチ (続)

- Matcher (Egison独自)
  - パターンマッチの方法を指定する

```
> (match-all {1 2 3 4} (list integer)
    [<cons $x $xs> [x xs]])
{[1 {2 3 4}]}

> (match-all {1 2 3 4} (multiset integer)
    [<cons $x $xs> [x xs]])
{[1 {2 3 4}] [2 {1 3 4}] [3 {1 2 4}] [4 {1 2 3}]}
```

Egisonのパターンマッチの特徴

1. 複数の結果を持つパターンマッチ
2. 非線形パターンマッチング
3. Matcherのモジュール化
4. パターンのモジュール化

複数の結果を持つパターンマッチ

- 集合のパターンマッチを扱うには複数の結果を持つパターンマッチを行えることが必須

```
> (match-all {1 2 3 4} (multiset integer)
    [<cons $x $xs> [x xs]])
{[1 {2 3 4}] [2 {1 3 4}] [3 {1 2 4}] [4 {1 2 3}]}

> (match-all {1 2 3 4} (list integer)
    [<join _ <cons $x <join _ <cons $y _>>>> [x y]])
{[1 2] [1 3] [2 3] [1 4] [2 4] [3 4]}
```

非線形パターンマッチ

- 同じパターン内に同じ変数を複数回使うことが可能

```
> (match-all {1 2 3 2} (multiset integer)
    [<cons $n <cons ,n _>> n])
{2 2}

> (take 6 (match-all primes (list integer)
           [<join _ <cons $n <cons ,(+ n 2) _>>>
            [n (+ n 2)]]))
{[3 5] [5 7] [11 13] [17 19] [29 31] [41 43]}
```

Matcherのモジュール化

- パターンマッチの方法もモジュール化可能
  - list, multiset, set
  - compact-list
  - mod
  - graph
- 集合以外も直接的にパターンマッチ可能！

パターンのモジュール化

- 頻出するパターンを再利用することが可能

```
> (define $twin
    (pattern-function [$pat1 $pat2]
      <cons (& $pat pat1)
       <cons ,pat
        pat2>>))
> (match-all {1 2 3 2} (multiset integer)
    [<cons $n (twin $t _)> [t n]])
{[2 1] [2 1] [2 3] [2 3]}
```

他の研究との比較

	Views [1]	Active Patterns [2]	First Class Patterns [3]	Egison
複数の結果のサポート	Not supported	Not supported	Perfect	Perfect
非線形パターン	Not supported	Partially supported	Not supported	Perfect
Matcherのモジュール化	Partially supported	Partially supported	Partially supported	Perfect

[1] P. Wadler. Views: A way for pattern matching to cohabit with data abstraction. In Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, page 313. ACM, 1987.
[2] M. Erwig. Active patterns. Implementation of Functional Languages, pages 21–40, 1996.
[3] M. Tullsen. First Class Patterns. Practical Aspects of Declarative Languages, pages 1–15, 2000.

私のプログラミング経験

- UNIXのシェル (2007年10月 – 2007年11月)
  - ターミナルを触り始めて2ヶ月でCを使ってシェルを実装しました
  - source : https://github.com/egisatoshi/egsh
- Schemeインタプリタ (2008年3月)
  - 初めてSchemeと関数型言語に触れたその日に、SchemeでSchemeインタプリタを実装し完成させました
- Schemeコンパイラ (2008年10月 – 2009年3月)
  - SchemeでSchemeコンパイラを作りました
  - source : https://github.com/egisatoshi/scheme-compiler
- Egison (2010年3月 – 現在)
  - Egisonを設計し、Haskellを使って実装しました
  - source : https://github.com/egisatoshi/egison3
- Ruby, Erlang, PostgreSQL, Emacs Lisp, OCaml, VHDL, Prolog, PHP, …

プログラミング言語Egisonのこれから

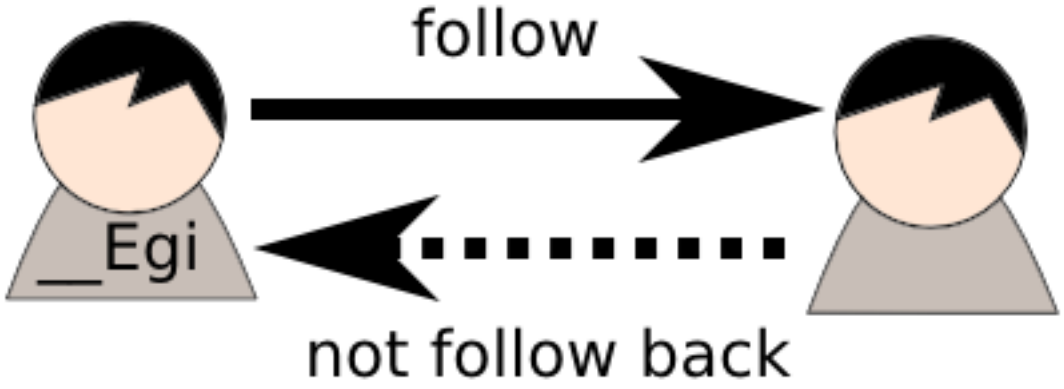
- Egisonのパターンマッチには非常に広い応用範囲がある
  - 強力なパターンの表現力を活かした大規模データ解析
  - プログラムの自動並列化による超高速計算

大規模データ解析

- データベースからのデータの取得は、集合のパターンマッチとして表現するのが自然
  - Egison Query Language
    - EgisonQLにより非常にシンプルにクエリを表現できる
      - » 長くて複雑なWhere節は必要ない
      - » サブクエリは必要ない
  - GraphDBのようなRDB以外のデータベース
    - まだSQLのようなスタンダードなクエリ言語がない
    - Egisonのパターンマッチで覇権を取りたい！

EgisonQLのデモ

- “__Egi”がフォローしているのに、“__Egi”をフォロー返ししてくれていないTwitterユーザー一覧を取得



SQL Version

- 複雑で理解するのが難しい
  - 長くて複雑なWhere節
  - サブクエリを含んでいる

```
SELECT DISTINCT ON (twitter_user4.screen_name) twitter_user4.screen_name
  FROM twitter_user AS twitter_user1,
       follow AS follow2,
       twitter_user AS twitter_user4
  WHERE twitter_user1.screen_name = '__Egi'
    AND follow2.from_uid = twitter_user1.uid
    AND twitter_user4.uid = follow2.to_uid
    AND NOT EXISTS
      (SELECT ''
         FROM follow AS follow3
         WHERE follow3.from_uid = follow2.to_uid
           AND follow3.to_uid = twitter_user1.uid)
  ORDER BY twitter_user4.screen_name;
```

EgisonQL Version

- 非常に簡潔
  - Where節がない！
  - サブクエリもない！

```
match-all [twitter_user follow follow twitter_user]
          [<cons (& <uid $uid> <screen_name ,"Egison_Lang">) _>
           <cons (& <from_uid ,uid> <to_uid $fid>) _>
           ^<cons (& <from_uid ,fid> <to_uid ,uid>) _>
           <cons (& <uid ,fid> <screen_name $sn>) _>]
  {<Uid fid> <Screen_name sn>}
```

分散並列計算

- 自動並列計算
  - Egisonは純粋関数型言語
  - 並列化可能なループはEgisonでは全てパターンマッチ式で記述される
  - Egisonのパターンマッチは自動並列実行可能
    - 詳細に興味を持って下さった方は、Egisonマニュアルの“Pattern-Matching Mechanism of Egison” の節を読んで頂けたら幸いです(http://egison.pira.jp/manual3/mechanism.html)
- プログラミング言語の抽象度が上がるほど、プログラムの解析が行い易くなり、最適化の可能性も広がる

参考文献

[1] P. Wadler. Views: A way for pattern matching to cohabit with data abstraction. In Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, page 313. ACM, 1987.

[2] M. Erwig. Active patterns. Implementation of Functional Languages, pages 21–40, 1996.

[3] M. Tullsen. First Class Patterns. Practical Aspects of Declarative Languages, pages 1–15, 2000.

終わり

- ご清聴ありがとうございました！