プログラミング言語Egison

萩谷研究室
江木聡志

発表の流れ

- イントロダクション
- サンプル
- アプリケーション
- 今後の課題

発表の流れ

- イントロダクション
  - Egisonの紹介
  - 関連研究
  - 貢献
- サンプル
- アプリケーション
- 今後の課題

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

既存のパターンマッチの問題点

- 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型(unfree data types)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！！

主な先行研究

- 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.
- M. Erwig. Active patterns. Implementation of Functional Languages, pages 21–40, 1996.
- M. Tullsen. First Class Patterns. Practical Aspects of Declarative Languages, pages 1–15, 2000.
- C. Queinnec. Compilation of non-linear, second order patterns on s-expressions. In Programming Language Implementation and Logic Programming, pages 340–357. Springer, 1990.

…

Non-free data types

- 同じデータに対して複数の表現形式があるデータ型
  - 複素数など
  - それぞれの表現形式の間の変換方法を定義することによって実現する
- Viewsなどの研究がある
  - 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.

```
complex ::= Pole real real

view complex ::= Cart real real
  in (Pole r t) = Cart (r * cos t) (r * sin t)
  out (Cart x y) = Pole (sqrt (x ^ 2 + y ^ 2)) (atan x y)

complex ::= Cart real real

view complex ::= Pole real real
  in (Cart x y) = Pole (sqrt (x ^ 2 + y ^ 2)) (atan x y)
  out (Pole r t) = Cart (r * cos t) (r * sin t)

add (Cart x y) (Cart x' y') = Cart (x + x') (y + y')

mult (Polar r t) (Polar r' t') = Polar (r * r') (t + t')
```

カルテシアン座標から，
極座標への変換方法を
定義している

極座標から，
カルテシアン座標への変換方法を
定義している

極座標による表現の入力が与えられても，
自動でカルテシアン座標による表現に変換さ
れてパターンマッチが行われる

Unfree data types

- 同じデータに同じ表現形式の中で複数の表現の方法があるデータ型
  - マルチセット、集合、リスト（Joinコンストラクタで定義した場合）など
  - パターンマッチの際に適宜、ターゲットの構造を変換しながらパターンマッチを行う
  - 複数の結果を扱えるパターンマッチの研究もある(First class patterns)

- Acitive patternsやFirst class patternsなどの研究がある
  - M. Erwig. Active patterns. Implementation of Functional Languages, pages 21–40, 1996.
  - M. Tullsen. First Class Patterns. Practical Aspects of Declarative Languages, pages 1–15, 2000.
- Non-free data typesのパターンマッチの研究はこれらの研究の基になった

```
data List a = Nil | Unit a | Join (List a) (List a)

cons x xs = Join (Unit x) xs

cons# Nil = Nothing
cons# (Unit a) = Just (a, Nil)
cons# (Join xs ys) = case cons# xs of
                       Just (x, xs') -> Just (x, Join xs' ys)
                       Nothing -> cons# ys
```

consパターンコンストラクタで,
joinで構成されたリストを分解する方法を定義している

この定義では、返り値にMaybe型が使われているが、
List型を用いれば、複数の結果を持つパターンマッチを実現できる

Non-linear patterns

- 同じパターンの中に、同じパターン変数が2つ以上現れるパターンも扱えるパターンマッチ
  - (?x ??- ?x)
  - (??x ??x)

- リストに対するnon-linearパターンマッチの研究はある
  - C. Queinnec. Compilation of non-linear, second order patterns on s- expressions. In Programming Language Implementation and Logic Programming, pages 340–357. Springer, 1990.

Egisonの貢献

- Unfree data typesに対するNon-linearなパターンマッチ機能を実現した
  - 複数の結果を扱えるパターンマッチをNon-linearなパターンにも対応できるようにした
- そのパターンマッチに探索を制御する仕組みを用意した

発表の流れ

- イントロダクション
- サンプル
  - Non-linearパターン
  - Cutパターン
  - ユークリッドの互除法
  - 型の定義
- アプリケーション
- 今後の課題

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

=> {2 7}
```

パターンの左側のパターン変数に束縛された値を参照できる

```
(test (match {5 2 1 3 4} (Multiset Integer)
        {[<cons $n
            <cons ,(- n 1)
              <cons ,(- n 2)
                <cons ,(- n 3)
                  <cons ,(- n 4)
                    <nil>>>>>>
          <ok>]
         [_ <ko>]}))

=> <ok>
```

パターンの左側で得られた束縛を環境に追加して評価される

```
(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
```

リストとしてパターンマッチを行うため，
一度ソートが行われる

{m, m, m, n, n}のように先頭3つが同じで残り2つが同じ場合と
{m, m, n, n, n}のように先頭2つが同じで残る3つが同じ場合

```
(test (match-all {{1 2 3 4 5} {4 5 1} {6 1 7 4}}
        (List (Multiset Integer))
        [<cons <cons $n _>
          <cons <cons ,n _>
           <cons <cons ,n _>
            <nil>>>>>
         n]))
```

ネストした型のパターンマッチも簡単に行える

```
=> {1 4}
```

```
(define $min
  (lambda [$ns]
    (match ns (List Integer)
      {[<cons $n <nil>> n]
       [<cons $n $rs>
        (let {[$r (min rs)]}
          (match (compare-integer n r) Order
            {[<less> n]
             [_ r]}))]})))

(define $gcd
  (lambda [$ns]
    (match ns (Set Integer)
      {[<cons $n <nil>> n]
       [<cons (& ,(min ns) $m)
              $rs>
        (gcd {m @((remove-all Integer)
                  (map (lambda [$r] (mod r m)))
                  rs)
                 0})]})))
```

minの中では整数のリストとしてパターンマッチが行われる

gcdの中では整数の集合としてパターンマッチが行われる

正規形を持たないデータは別々の場所でそれぞれ別々の型として扱われることが多い

Egisonの型の定義

- パターンとデータを引数に取り，全ての可能な束縛を計算するマッチ関数なるものを型ごとにユーザは記述する
  - 例．Multisetのマッチ関数は．パターン<cons $x $xs>，ターゲット{1 2 3}の場合，以下のような結果を返す
    - `{[x 1] [xs {2 3}]},{[x 2] [xs {1 3}]},`
      `{[x 3] [xs {1 2}]}`
- マッチ関数の記述のために以下の関数を定義する
  - Var-match: パターンが変数パターンである場合にパターンマッチを行う関数
  - Inductive-match: パターンが帰納的に構成されたパターンである場合にパターンマッチを行う関数

```
(define $Multiset
  (lambda [$a]
    (type
      {[$var-match (lambda [$tgt] {tgt})]
       [$inductive-match
        (deconstructor
          {[nil []
            {[{} {[]}]
             [_ {}]}]
           [cons [a (Multiset a)]
            {[$tgt (map (lambda [$t] [t ((remove a) tgt t)])
                        tgt)]}]
           [join [(Multiset a) (Multiset a)]
            {[$tgt (map (lambda [$ts] [ts ((remove-collection a) tgt ts)])
                        (subcollections tgt))]}]})]
       [$equal? (lambda [$val $tgt]
                  (match [val tgt] [(Mult
                    {[[<nil> <nil>] <true>]
                     [[<cons $x $xs> <cons ,x ,xs>] <true>]
                     [[_ _] <false>]}))]})))
```

nilのためのinductive-match

consのためのinductive-match

joinのためのinductive-match

発表の流れ

- イントロダクション
- サンプル
- アプリケーション
  - ポーカー
  - グラフ
  - 電子回路
- 今後の課題

アプリケーション

- ポーカーの役判定
  - 集合のパターンマッチが直接表現できるおかげで簡潔に記述できる
  - 麻雀も簡単に書ける
- グラフのパターンマッチ
  - グラフはノードとエッジの多重集合として表現できる
  - 電子回路の変換
    - 端子への入力は多重集合として表現できる
- …

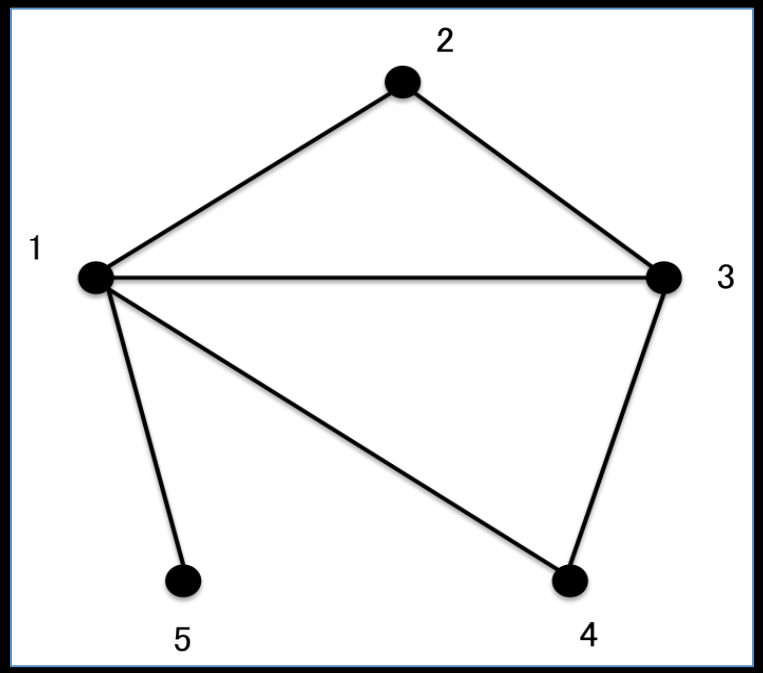
```
(define $Node Integer)

(define $NodeInfo
  (type
    {[$var-match (lambda [$tgt] {tgt})]
     [$inductive-match
      (deconstructor
        {[node [Node (Multiset Node) (Multiset Node)]
          {[<node $n $in $out> {[n in out]}]}]})]
     [$equal? (lambda [$val $tgt]
                (match [val tgt] [NodeInfo NodeInfo]
                  {[[<node $n $in $out> <node ,n ,in ,out>] <true>]
                   [[_ _] <false>]}))]}))

(define $Graph (Multiset NodeInfo))
```

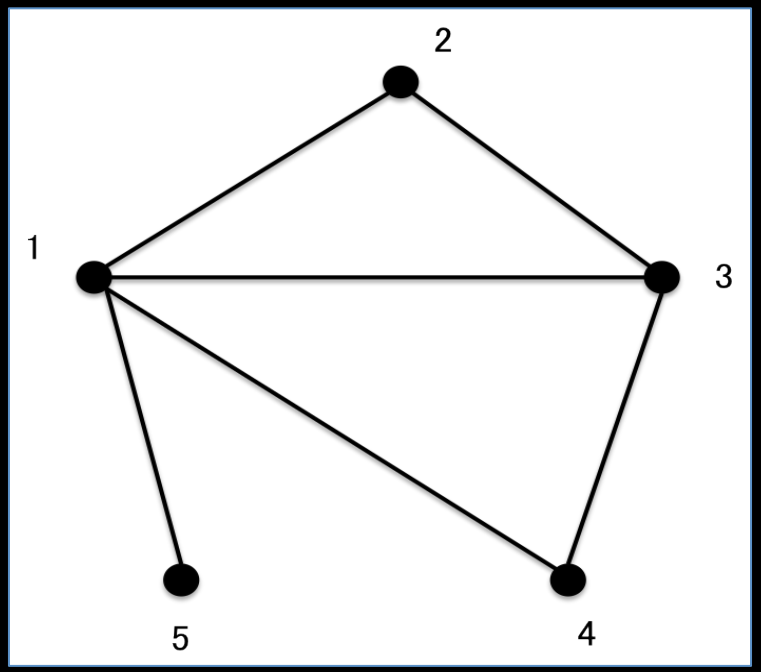
Node Id, 入ってくるエッジの先のノード，出ていくエッジの先のノード

```
(define $g {<node 1 {2 3 4 5} {2 3 4 5}>
            <node 2 {1 3} {1 3}>
            <node 3 {1 2 4} {1 2 4}>
            <node 4 {1 3} {1 3}>
            <node 5 {1} {1}>})
```



```
(test (match-all g Graph
        [<cons <node $n1 <cons $n2 _> _>
          <cons <node ,n2 <cons $n3 _> _>
           <cons <node ,n3 <cons ,n1 _> _>
             _>>>>
         [n1 n2 n3]]))
```

3つの辺からなる閉路にマッチするパターン




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

発表の流れ

- イントロダクション
- サンプル
- アプリケーション
- 今後の課題

今後の課題

- パラメーターの値によってパターン変数の個数が変化するパターンの記述
  - グラフのパターンマッチの場合
    - ハミルトン閉路、巡回セールスマンのパターンマッチがひとつのパターンで表現できるようになる

- 多次元データに対する直感的なパターンマッチ
  - 碁盤や将棋盤のわかりやすいパターンマッチを行いたい
  - ここまでできたら「ノーベル賞級の功績」by竹内先生

- コンパイラの作成
  - 未踏に採択された！「プログラミング言語Egisonのコンパイラの開発」

今後の課題

- パラメーターの値によってパターン変数の個数が変化するパターンの記述
  - グラフのパターンマッチの場合
    - ハミルトン閉路、巡回セールスマンのパターンマッチがひとつのパターンで表現できるようになる
  - 「この拡張はMUST」by竹内先生

- 多次元データに対する直感的なパターンマッチ
  - 碁盤や将棋盤のわかりやすいパターンマッチを行いたい

- コンパイラの作成
  - 未踏に採択された！「プログラミング言語Egisonのコンパイラの開発」

今後の課題

- パラメーターの値によってパターン変数の個数が変化するパターンの記述
  - グラフのパターンマッチの場合
    - ハミルトン閉路、巡回セールスマンのパターンマッチがひとつのパターンで表現できるようになる
  - 「この拡張はMUST」by竹内先生

- 多次元データに対する直感的なパターンマッチ
  - 碁盤や将棋盤のわかりやすいパターンマッチを行いたい
  - ここまでできたら「ノーベル賞級の功績」by竹内先生

- コンパイラの作成
  - 未踏に採択された！「プログラミング言語Egisonのコンパイラの開発」

今後の課題

- パラメーターの値によってパターン変数の個数が変化するパターンの記述
  - グラフのパターンマッチの場合
    - ハミルトン閉路、巡回セールスマンのパターンマッチがひとつのパターンで表現できるようになる
  - 「この拡張はMUST」by竹内先生

- 多次元データに対する直感的なパターンマッチ
  - 碁盤や将棋盤のわかりやすいパターンマッチを行いたい
  - ここまでできたら「ノーベル賞級の功績」by竹内先生

- コンパイラの作成
  - 未踏に採択された！「プログラミング言語Egisonのコンパイラの開発」
  - 「これは必要ない」by竹内先生