コレクションデータに対してのパターンマッチの実現

江木聡志 (東京大学)

理念

- アルゴリズムの直感的な表現を目指している
- 計算機向けに一切の翻訳なしに，アルゴリズムに対する人間の　イメージをそのまま記述できる言語

コレクションデータとは

- 抽象データの一種
- 任意個の同じ種類のデータの組み合わせからなるデータ
- 例えば，リストや多重集合(マルチセット)，集合など

問題点

- リスト以外のコレクションデータに対するデコンストラクタの記述は煩雑
- プログラマは多くの場合，リスト以外のコレクションデータについて，それをリストとしてデコンストラクトできるように，処理を無理やり捉え直す
- 正規形をもたないことが原因
  - { a, b, c } = { b, a, c } = { c, b, a } = { a, a, b, c } (集合としてみた場合)

目標

- コレクションデータに対するデコンストラクトのアルゴリズムは，人間にとっては単純
- 集合は集合として，多重集合は多重集合として直接扱いたい

本研究の提案

- コレクションデータ専用のコンストラクタの用意
- コレクションデータ専用のデコンストラクタの定義方法を用意

- 従来の方法では，コレクションデータは他の抽象データと区別されず，型の直和，直積，再帰を用いて定義され，その定義にもとづきコンストラクタ，デコンストラクタも定義されていた

発表の流れ

- 本研究のアイデアを組み込んだ言語の構文の説明
- 本研究の言語によるプログラムの例
- コレクションデータのパターンマッチのアルゴリズムの説明
- コレクションデータのマッチ関数（デコンストラクタ）の記述の例
- 関連研究との比較

発表の流れ

- **本研究のアイデアを組み込んだ言語の構文の説明**
- 本研究の言語によるプログラムの例
  - コレクションデータのパターンマッチのアルゴリズムの説明
  - コレクションデータのマッチ関数（デコンストラクタ）の記述の例
  - 関連研究との比較

構文（全体）

```
<exp> ::= <con>                                    (定数)
        | <var>                                    (変数)
        | (<exp> <exp>)                            (関数適用)
        | (lambda (<var> ...) <exp>)
                                                   (関数定義)
        | (let ((<var> <exp>) ...)
            <exp>)                                 (let式)
        | (collect <c-exp> ...)
            (コレクションデータのコンストラクタ)
        | <match-exp>                              (マッチ式)
```

構文（コレクションデータ）

<exp> ::= (collect <c-exp> ...)

<c-exp> ::= <exp>　　(要素)

| .<exp>　　(サブデータ)

例.

(collect 1 2 3) => {1 2 3}

(collect 1 2 .(collect 3 4) 5) => {1 2 3 4 5}

構文（マッチ式）

```
<match-exp> ::=                                    (マッチ式)
    (match <target> <match-function>
        (<condition> <exp>)
        ...)

<target> ::= <exp>                                 (ターゲット)

<condition> ::=                                    (条件)
    (<pattern> (<rule> ...))
```

マッチ関数（型）の例

```
Int                                    (整数のマッチ関数)
(** Int String)                        (整数と文字列の直積)
(|| Int String)                        (整数と文字列のユニオン)

(define Bintree (lambda (inner-type)
  (|| inner-type
      (** inner-type
          (Bintree inner-type)
          (Bintree inner-type)))))

(Bintree Int)                                (整数の二分木)

(Multiset Int)                               (整数の多重集合)
(Set (Set Int))                              (整数の集合の集合)
```

構文（パターン）

```
<condition> ::= (<pattern> (<rule> ...))

<pattern> ::= <var>                    (変数パターン)
            | _                        (ワイルドカード)
            | ,<exp>                   (定数パターン)
            | (<pat-exp> ...)
                                       (コレクションパターン)

<pat-exp> ::= <pattern>                (要素パターン)
            | .<pattern>               (サブデータパターン)
```

例（パターン）

パターン `(a ,5 _ .S c)` に対して

ターゲット `(collect 6 5 4 3 2 1)` を

リストとしてパターンマッチしたとすると,

```
a = 6,
S = (collect 3 2),
c = 1
```

という束縛を返す.

構文（ルール）

```
<condition> ::= (<pattern> (<rule> ...))

<rule> ::= (= <var> (<var> ...) <exp>)
         | (: <var> (<var> ...) <exp>)
         | (? <var> (<var> ...) <exp>)
```

例.

```
(= n (x y) (+ x y))          : n = x + y
(: n () (collect 1 2 3))     : n ∈ {1, 2, 3}
(? n () even?)               : nは偶数
```

発表の流れ

- 本研究のアイデアを組み込んだ言語の構文の説明
- **本研究の言語によるプログラムの例**
  - コレクションデータのパターンマッチのアルゴリズムの説明
  - コレクションデータのマッチ関数（デコンストラクタ）の記述の例
  - 関連研究との比較

プログラムの例

```
(define gcd
  (lambda (Ns)
    (match Ns (Set Int)
      (((m ,0) ()) m)
      (((m .Rest)
        ((= m () (min (remove Ns 0)))))
       (gcd (collect m
                     .(map (lambda (n)
                             (modulo n m))
                           Rest)))))))
```

```
(define poker_hands
  (lambda (Cs)
    (match Cs (Multiset (** Mark Mod13))
      ((((S ,10) (S ,11) (S ,12) (S ,13) (S ,1))
        ())
       "Royal Straight Flush")
      ((((S n) (S n+1) (S n+2) (S n+3) (S n+4))
        ((= n+1 (n) (+ n 1))
         (= n+2 (n) (+ n 2))
         (= n+3 (n) (+ n 3))
         (= n+4 (n) (+ n 4))))
       "Straight Flush")
      ((((_ n) (_ n) (_ n) (_ n) _)
        ())
       "Four of Kind")
      ((((_ m) (_ m) (_ m) (_ n) (_ n))
        ())
       "Full House")
      ((((S _) (S _) (S _) (S _) (S _))
        ())
       "Flush")
```

```
((((_ n) (_ n+1) (_ n+2) (_ n+3) (_ n+4))
  ((= n+1 (n) (+ n 1))
   (= n+2 (n) (+ n 2))
   (= n+3 (n) (+ n 3))
   (= n+4 (n) (+ n 4))))
 "Straight")
((((_ n) (_ n) (_ n) _ _)
  ())
 "Three of Kind")
((((_ m) (_ m) (_ n) (_ n) _)
  ())
 "Two Pair")
((((_ n) (_ n) _ _ _)
  ())
 "One Pair")
(((_ _ _ _ _)
  ())
 "Nothing"))))
```

発表の流れ

- 本研究のアイデアを組み込んだ言語の構文の説明
- 本研究の言語によるプログラムの例
- **コレクションデータのパターンマッチのアルゴリズムの説明**
- コレクションデータのマッチ関数（デコンストラクタ）の記述の例
- 関連研究との比較

コレクションデータのパターンマッチの
アルゴリズム（概要）

<pattern> ::=(<pat-exp> ...)　（コレクションパターン）

<pat-exp> ::= <pattern>　（要素パターン）

| .<pattern>　（サブデータパターン）

コレクションパターンのそれぞれの要素のとる値を一つずつ確定していき，全ての要素について値が確定したらパターンマッチ終了となります.

コレクションデータのパターンマッチのアルゴリズム

- コレクションパターンのそれぞれの要素の取りうる値の集合（**マッチ域**）を計算

- 取りうる候補の少ない要素から暫定的に確定していく
    - もし，取りうる候補がない要素があったら，前に暫定的に確定した要素にもどり，バックトラックする．そのような要素のない場合はパターンマッチ失敗

- 全ての値が確定したらパターンマッチ完了

マッチ域の計算の流れ

- コレクションパターンのそれぞれの要素について，ルールを考慮せずに，取りうる値の集合（**可動域**）を計算する．
- コレクションパターンのそれぞれの要素について，それをパターン，可動域の要素をターゲットとして，パターンマッチできるか調べる．パターンマッチに成功した可動域の要素の集合がマッチ域となる．

パターンマッチの例

マッチ関数（型）: (Multiset Int)

ターゲット　　: (2 3 6)

パターン　　: (a b c)

ルール　　: ((? a () even?) (= b () 6))

としてパターンマッチを行う.

パターンマッチの例（2）

パターン ： (a b c)

についてそれぞれ可動域を計算する.

a, b, c 全てについて可動域は, {2, 3, 6}である.

次に, a, b, c それぞれについてマッチ域を計算する.

パターンマッチの例（3）

まず，a についてマッチ域を計算する.

マッチ関数 : Int

パターン : a

ルール : ((? a () even?) (= b () 6))

とし，{2，3，6} のそれぞれをターゲットとして

パターンマッチするか調べる.

2，6 についてパターンマッチに成功する.

コレクションパターン (a b c) の要素，a のマッチ域

は {2，6} となる.

パターンマッチの例（４）

同様にして，`b` のマッチ域は `{6}`，`c` のマッチ域は `{2，3，6}` とわかる.

マッチ域が一番少ないコレクションパターンの要素は，`b` なので，`b` が `6` に確定される.

パターンマッチの例（５）

b を 6 に確定し，

パターン ：（a 6 c）

として，再度パターンマッチを行う．

a，c についてそれぞれ可動域を計算する．

a，c どちらについても，可動域は {2，3}である．

先程とどうように再帰的にパターンマッチを行い

マッチ域を計算する．

a のマッチ域は {2}，c のマッチ域は {2，3} となる．

パターンマッチの例（６）

マッチ域の要素が一番少ない a を 2 に確定する.

パターン : (2 6 c)

として再度パターンマッチを行う.

そして, c = 3 とすればマッチ可能という結論を得る.

コレクションデータのパターンマッチのアルゴリズム（まとめ）

- この方法で全ての種類のコレクションデータのパターンマッチを行う
- コレクションデータの種類によって異なるのは可動域の計算の部分だけ
- 人間がコレクションデータのパターンマッチを行う際のアルゴリズムとほぼ同じ

発表の流れ

- 本研究のアイデアを組み込んだ言語の構文の説明
- その言語によるプログラムの例
- コレクションデータのパターンマッチのアルゴリズムの説明
- **コレクションデータのマッチ関数（デコンストラクタ）の記述の例**
- 関連研究との比較

```
(define Multiset
  (match-function (pattern target inner-type)
    (let* ((cs (filter fixed? pattern))
           (rs (remove-all inner-type
                           target cs)))
      (movable-region pattern
        ((((.Fs1) u (.Fs2))
          ((? Fs1 () all-fixed?)
           (? u () unfixed?)
           (? Fs2 () all-fixed?)))
         (if (= (length rs) 1)
             (collect .rs)
             (collect)))
        ((((._) u (._))
          ((? u () unfixed?)))
         (collect .rs))
        ...
```

```
...
(((((.Fs1) S (.Fs2)))
  ((? Fs1 () all-fixed?)
   (? S () subdata?)
   (? Fs2 () all-fixed?)))
 (collect rs))
(((((.FSs1) S (.FSs2)))
  ((? Fs1 () all-fixed-or-subdata?)
   (? S () subdata?)
   (? Fs2 () all-fixed-or-subdata?)))
(collect .(submultisets rs))))))))
```

実装

- 本研究の理論に基づいたプロトタイプ実装
  - 先程の例の gcd のパターンマッチや，ポーカーの役判定のパターンマッチ，他にも集合の集合のパターンマッチが動くことを実証済み

発表の流れ

- 本研究のアイデアを組み込んだ言語の構文の説明
- その言語によるプログラムの例
- コレクションデータのパターンマッチのアルゴリズムの説明
- コレクションデータのマッチ関数（デコンストラクタ）の記述の例
- **関連研究との比較**

Active Patterns [Erwig. 1997] との比較（１）

- 抽象データについてのパターンマッチの方法をプログラマが指定できる
- ただしコレクションデータを他の抽象データと区別していない
  - コレクションデータも他の抽象データと同じく型の直積，直和，再帰を用いて定義する

Active Patterns [Erwig. 1997] との比較（２）

- それゆえ以下のような制限がある
  - パターンの先頭から順番に確定していかなければならない
  - バックトラッキングを行うことを考えていない

結論

- コレクションデータを他の抽象データと区別した
    - 専用のコンストラクタと，バックトラッキングを組み込んだ専用のデコンストラクタを用意
- これにより多くのアルゴリズムの記述が直接的になった

今後の課題

- より洗練されたマッチ関数の記述方法の考案
- 関連研究の調査
  - Logic Programming との表現力や効率の差