神奈川大学理学部　情報科学科　馬谷研究室

# Racket 言語の抽象構文木パターンマッチ機能の改善

今村 康平 201703339

## 1　はじめに

普通の関数でエラーを検出するには，エラーをチェックするコードを導入する他に, 契約や, 静的型検査を使う．マクロでも，契約や型に相当するチェック方法がある．それが syntax-parse [2] であり，構文パターンの型に相当するのが，syntax-class である．syntax-parse は，クラスによって型が決まり，そのクラスかどうかをパターンマッチ機能を用いてエラーを検出していく．

しかしながら，パターンマッチ機能の実行効率については不十分な箇所が存在し, 改善の余地がある．そこで本研究では，パターンマッチ機能の改善方法を明らかにする．これによって，syntax-parse を使用するプログラムの実行を高速化できる可能性がある．

本研究では，2 種類の方法で改善した．1 つ目に，重複したクラスがパターンマッチした際の動作を改善した．2 つ目に，一貫性のないクラスをパターンマッチした際の動作を改善した．

第 2 節では，研究背景について説明する．第 3 節では，提案する改善手法について説明する．第 4 節では，実際に実装する．第 5 節では，まとめとして本研究の総括を行う．

## 2　背景

マクロを書くためのパターンマッチを行う構文には，syntax-rules，syntax-case など様々存在する．その中でも特に syntax-parse [2] と呼ばれる高機能なパターンマッチ構文が存在する．syntax-parse では，構文パターンの「型」を表す syntax-class を用いることで文法の種類を区別し，文法エラーのチェックを自動で行ってくれる．型の種類として，id（識別子），number（数）などが例に挙げられる．

以下に syntax-parse の簡単な利用例を示す．

```
(define-syntax (foo stx)
  (syntax-parse stx
      [(_ x:number) #''x]))
```

syntax-parse は，foo の引数を受け取り，その引数が number クラスかどうかパターンマッチを行う．(foo 1) と実行すると，1 が返され，(foo a) とすると，エラーが返される．

ここで問題点が二つある．一つ目は，同じクラスを持った 2 つのパターン変数を syntax-parse で実行すると, パターンマッチを 2 回行うことである．もし，同じクラスを持っていたら，1 回のパターンマッチのみに省略することはできないだろうか．二つ目は，id を持ったパターン変数をパターンマッチした後，その変数を number でもパターンマッチさせるような，一貫性のないパターンマッチを行っていることである．再度のパターンパッチを試みる前にエラーで表示できないだろうか．本研究では，このようなパターンマッチ機能の改善を行う．

## 3　改善方針

例えば，重複しているパターンマッチを以下のコードで考える．

```
(define-syntax (bar stx)
  (syntax-parse stx
    [(_ x:myid)
     #:with y:myid #'x
     #''y]))
```

1 回目のクラス myid のパターンマッチを行い，

4 行目の際に，同じ myid クラスを持っているのであれば，このパターンマッチでの型検査は冗長であり省略できる．それを可能にするために，syntax-property を使用する．この変数パターンにマッチするコードにプロパティという付加情報を付ける．この情報があることにより，1 回目の myid でパターンマッチしたかどうか判断が可能になる．

次に，一貫性のないパターンマッチを以下のコードで考える．

```
(define-syntax (foo stx)
  (syntax-parse stx
    [(_ x:id)
     #:with n:number #'x
     #'n]))
```

id を持った変数をパターンマッチした後，その変数を number でもパターンマッチしたら，コンパイルしたと同時に，エラーを出すようなコードを作る．これを可能にするためにも先程と同様，syntax-property を使用する．id のクラスを持ったパターン変数をさらに，number でもパターンマッチさせることを防ぐことができる．

## 4 実装

まず，改善を実現するために，新しい syntax-parse を用意した．これを syntax-ima-parse とした．

```
(syntax-ima-parse stx
  [(foo x:myid)
   (syntax-ima-parse #'x
     [y:myid #''y])]
  [(foo y) #'1])
```

以降に書かれているような処理を行うコード（図 1）へとマクロにより変換する．x がプロパティを持っているか判断するために，syntax-property を用いて分岐させる．プロパティを持っているなら，そのクラスは myid かどうか判断し，myid だったら，1 回目のパターンマッチを行う．持っていないなら，syntax-parse で x がクラス myid であるか，それ以外かで 1 回目のパターンマッチを行い，myid だったら，x の class に'myid というプロパティを付けて，2 回目のパターンマッチを行う．

```
(syntax-parse stx
  [(foo x)
   #:do [(define result
   (if (syntax-property #'x 'class)
    (if (x のプロパティが myid である)
                    'success1, #f)
    (if (x がクラス myid を持っている)
                    'success2, #f)))]
   #:when result
    (cond ((eq? result 'success1)
             パターンマッチ)
          ((eq? result 'success2)
            'myid を付けた後,
                   パターンマッチ))
  ]
  [(foo y)  #''y])
```

図 1　変換後のソースコード

## 5 まとめ

本研究では，Racket 言語の構文パターンマッチ機能の改善を行った．重複したクラスや一貫性のないクラスのパターンマッチを改善することで，効率よくパターンマッチを行えるようになった．

## 参考文献

[1] Fear of Macros. http://www.greghendershott.com/fear-of-macros/

[2] Culpepper, R. Fortifying macros, Journal of Functional Programming, Vol.22, pp.439–476, DOI: https://doi.org/10.1017/S0956796812000275, 2012.