神奈川大学理学部　情報科学科　馬谷研究室

Racket の構文解析機能 syntax-parse の効率化

有田 健一郎 201803407

1　はじめに

Racket とは，Scheme から派生したプログラミング言語である．この Racket には syntax-parse という構文解析のための機能が備わっており，syntax-parse を使用するマクロは，マクロの構文パターンに埋め込まれた記述やメッセージに基づいて，自動的に理解しやすいエラーメッセージを生成することができる．そのようなエラーメッセージを生成するためのパターン中の記述方法の一つに，構文パターンの「型」を指定する syntax-class がある．

しかしながら，syntax-class に関わるマッチング処理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，syntax-class により指定された「型」を検査する処理の改善方法を提案する．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 種類の方法で改善を行った．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．

2 節では，研究背景について説明する．3 節では，提案する改善手法について説明する．4 節では，提案手法の実装について述べる．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 などいくつか存在する．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．典型的な「型」の例として，id（識別子），number（数），expr（式）などが挙げられる．

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，マクロ呼び出し式全体を表す stx をパターンマッチ対象とし，呼び出し式の第 1 引数が number クラスであるかどうかパターンマッチを行う．たとえば (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可能である．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が可能である．本研究では，これら 2 点においてパターンマッチ機能の改善を行う．

3　改善方針

まず初めに，重複しているパターンマッチの改善方法として考えられるのは，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とである．この情報があることにより，1 回目のパターンマッチをしたかどうかの判断が可能になる．この方法は syntax-property を使用することで実

現が可能である．この syntax-property を使うことで，パターンマッチの対象である構文オブジェクトに対して，情報が付加されているかの判断を行い，さらに，付加されていないのであれば，パターンマッチの際に付加させるように利用する．次に，たとえば 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についても改善する．syntax-parse をもう一度行い，構文を解析していき，コロン記法ではない方法でパターンマッチを行う，#:declare を使用することで改善できる．

4 実装

まず，既存の syntax-parse の機能を流用しながら改善手法を実現するために新しい syntax-parse を用意した．以降，改善手法を実現している syntax-parse を fast-syntax-parse と呼ぶことにする．

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

図 1 変換前のソースコード

図 1 の例を使って説明する．図 1 を，以降に書かれているような処理を行うコード（図 2）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，if を用いて分岐させる．プロパティを持っているなら，result に'success1 という値を，プロパティを持っていないなら，syntax-parse で x がクラス myid であるか検査する．この時点でプロパティを持っておらず，かつ，x のクラスが myid でもなかった場合，一貫性のないパターンマッチに該当するため，エラーコードを表示させる．

さらに，プロパティを持っている，または，クラスが myid であった場合の処理を#:when 以降に記述する．まず，プロパティを持っていた場合はそのままパターンマッチし，プロパティを持っていないが，クラスが myid であった場合には with-syntax を使い，新たに x に対してプロパティを与えた構文を x2 としてパターンマッチさせる．

```
(syntax-parse stx
  [(_ x)

   #:do [(define result
           (if
            (eq?
             (syntax-property #'x 'class)
             'myid)
            'success1
            (syntax-parse #'x
              [x
               #:declare x myid
               'success2])))]

   #:when result
   (cond
     ((eq? result 'success1)
      #'(fast-syntax-parse x
                           [y:myid #'y]))
     ((eq? result 'success2)
      (with-syntax
          ([x2 #'(syntax-property
                  #'x
                  'class
                  'myid)])
        #'(fast-syntax-parse x2
                             [y:myid #'y]))))
   ]
  [(_ y) #'1])
```

図 2 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効率化を行った．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，効率良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] Racket Documentation/Syntax: Meta-Programming Helpers/1.3 Parsing Syntax. https://docs.racket-lang.org/syntax/Parsing_Syntax.html

[3] 今村康平，Racket 言語の抽象構文木パターンマッチ機能の改善，2020 年度神奈川大学理学部情報科学科卒研要旨集，2021.