神奈川大学理学部　情報科学科　馬谷研究室

Rossete の生成文法への繰り返し構文の追加の検討

木村 匡 201803371

1　はじめに

Rosette とは，Scheme から派生したプログラミング言語である Racket のサブセットを拡張したドメイ特化言語である．

Rosette には，ユーザー自身が BNF 記法や正規表現なようなものを実際に Rosette の構文を使うことによって，生成文法へ定義することができる．構文には choose などがあるが，要素の繰り返しの処理を行うような構文がない．

そこで，探索空間の縮小や，生成されるコードの形を限定する repeat という要素の繰り返しを行う構文の追加の検討をすることを目的として本研究を進める．

2　Rosette

Rosette とは，シンボリック値，アサーション，仮定，およびクエリという 4 つの重要な概念に基づいていており，Racket とは異なる点である．

また，Rosette は次の 2 つのコンポーネントを備えたソルバー支援プログラミングシステムである．1 つ目は，Racket のサブセットを拡張し，制約解消系にアクセスするためのコンストラクトを備えたプログラミング言語である．このソルバーの助けを借り，Rosette はバグがあるか，ある場合はどのように修復するかなどのプログラムに関する興味深い質問に答えることができる．2 つ目は，Rosette のプログラムを実行し論理制約にコンパイルするシンボリック仮想マシン（SVM）である．この SVM により，Rosette はソルバーを使ってプログラムの動作を自動的に推論することができる．

3　生成文法

本研究の目的は，Rossete の生成文法への繰り返し構文の追加の検討をすることである．

Rosette には，ソルバーの入力となるコードの断片を自動生成する機能が備わっており，生成するコードの文法を定義するコンストラクトが提供されている．

Rosette では生成文法を以下のように記述することができる．

```
(define-grammar (fast-int32 x y)
  [expr
   (choose x y (?? int32?)
           ((bop) (expr) (expr))
           ((uop) (expr)))]
  [bop
   (choose bvadd bvsub bvand
           bvor bvxor bvshl
           bvlshr bvashr)]
  [uop
   (choose bvneg bvnot)])

; <expr> := x | y | <32-bit integer constant> |
;           (<bop> <expr> <expr>) |
;           (<uop> <expr>)
; <bop>  := bvadd  | bvsub | bvand |
;           bvor   | bvxor | bvshl |
;           bvlshr | bvashr
; <uop>  := bvneg | bvnot
```

これは，例題として公式サイト [1] に記載されたビットベクトルの中点計算問題に使われた生成文法である．

<expr>とは，実行結果を出せるように定義，<bop>とはビットベクトルの計算に必要な演算子を定義，<uop>ではオーバーフローした時のため，ビットベクトルの算術演算子を定義してある．

実際に，自動生成されたコードを以下に示す．

```
(define (bvmid-fast lo hi)
```

```
  (fast-int32 lo hi #:depth 2))
(define sol
    (synthesize
     #:forall    (list l h)
     #:guarantee (check-mid bvmid-fast l h)))
sol
(model
  ...)
(list
 'define
 '(bvmid-fast lo hi)
 (list 'bvlshr
       '(bvadd hi lo) (bv #x00000001 32)))
```

synthesize の中で条件を書くことでその条件を満たすコードを自動生成し，sol に保存する．sol を呼び出すと生成されたコード使い，自動で結果を出すことができる．

4 repeat 構文

本研究では，生成文法に要素の繰り返しを表す repeat 構文を追加の検討を行うことが目的である．要素の繰り返しの処理を行う構文を (repeat X n) とコードを書く．このコードは非終端記号 X を n 回繰り返すことを意味する．拡張する利点としては，探索空間の縮小や，生成されるコードの形を限定できる点が挙げられる．

5 repeat 構文の応用

repeat 構文を実際に使った生成文法を検討し，その例を下記に示す．

```
(define-grammar (call-n n)
    [call
        ((lam n) (repeat (expr) n))]
    ; ラムダ式に n 回増やした (expr) を渡す
    [(lam n)
        (lambda ((repeat (ids) m)) (expr))]
    ; ラムダ式の定義
    [ids
        (let (x (repeat (??) m)) (expr))]
    ; 生成されたものを保存
    [expr
        (choose
            ((bop) (expr) (expr)))]
    ; 処理の決定
    [bop
        (choose "+" "-" "*" "/" )])
    ; 演算子の定義
```

上記では，簡単な計算を行うための生成文法を定義した．call でラムダ式に処理は (expr) を渡すため，(lam n) では repeat を使ったラムダ式を，expr では実行結果を出すため，bop では四則演算を行えるよう，演算子を定義している．

```
(define-grammar (call-map n m)
    [call
        (map (lam n) (repeat (list m) n))]
    ;map 関数の定義
    [(lam n)
        (lambda ((repeat (ids) m)) (expr))]
    ; ラムダ式の定義
    [(list m)
        (repeat (list (ids)) m)]
    ; リストの中身を増やす
    [ids
        (let (x (repeat (??) m)) (expr))]
    ; 生成されたものを保存
    [expr
        (choose
            ((bop) (expr) (expr)))]
    ; 処理の決定
    [bop
        (choose "+" "-" "*" "/" )])
    ; 演算子の定義
```

上記では，リスト複数個使った計算を行うための生成文法を定義した．call では map 関数を呼び出すため，(lam n) では repeat を使ったラムダ式を，(list m) では list の中の要素を m 回増やすため，expr では実行結果を出すため，bop では四則演算を行えるよう，演算子を定義している．

実際のコードで考えるために，2 つ目の文法が使えない例を下記に示す．

```
(map (lambda (x y) (+ x y)))

'(1 2 3 4)
'(10 20 30 40 50)
```

上記のコードは，リストの数は一致しているがリストの要素数は一致していない．このような場合では計算を行っても，ユーザーが期待している実行結果を返すことができない．

次に，使える例を下記に示す．

```
(map (lambda (x y) (+ x y)))

'(1 2 3 4 5)
'(10 20 30 40 50)
=> '(11 22 33 44 55)
```

上記のコードは，実際に生成文法を使って計算された例である．ここではリストの数，リストの要素数が一致しているため，実行を行うことができる．

6 まとめ

本研究では，Rosette の生成文法への繰り返し構文の追加の検討を行った．提案した repeat 構文を用いることで，探索空間を縮小することができるだろう．

参考文献

[1] The Rosette Guide, Emina Torlak https://docs.racket-lang.org/rosette-guide, January 21, 2022.