神奈川大学理学部 情報科学科 馬谷研究室

遷移系記述言語における束縛を表現する機能の HOAS による実現

下山 桃佳 202104691

1 はじめに

遷移系記述言語 LwRed (Light Weight REDucer)[1] とは，実システムに組み込むことを目的とした遷移系を記述するために，拡張性と効率性の両方の利点を備えた領域特化言語である．本研究では, 先行研究 [1] 内で純粋 λ 計算の簡約を表現する際に定義が必要となる，代入を表す関数 `subst` に着目する．純粋 λ 計算より複雑な束縛を持つ構文の定義する際に，`subst` をユーザが正しく定義する必要があるため，LwRed において任意の束縛を表現することが可能な構文を定義する機能の実現を目標とする．実現にあたり, HOAS (Higher-Order Abstruct Syntax)[2] という手法を利用した `define-term` というマクロを実装することで，場合によっては複雑になりがちなプログラム中の束縛関係の表現を実現する．

2 背景

先行研究 [1] において，束縛の発生する λ 計算のような構文を定義する際には補助関数である `subst` を定義することで簡約を表現していた．しかし，その他のより複雑な束縛を含む構文を定義する場合，`subst` も複雑な束縛関係を間違いなく処理できるように拡張しなくてはならない．故に，本研究で実現を目標とする束縛を表現する機能に HOAS[2] という手法を利用することで，より複雑な束縛の表現を可能とし，`subst` のような補助関数を定義することなく，任意の構文を処理できる機能の実現を図る．

HOAS とは，通常の抽象構文木のデータ型を一般化したものであり，構文オブジェクトを表現するために使用される対象言語の束縛を，ホスト言語の束縛で表す手法である．本研究では束縛を含む構文を定義するための機能中に HOAS を利用するため，ホスト言語である Racket[3] において局所的な変数や関数を定義する `lambda` を用いる．

以下に HOAS を用いた
$((\lambda\ x.x+1)(2))$ の β 簡約を定義する例を示す:

```
(match `(λ ,(lambda (x) (+ x 1)) 2)
    [ `(λ ,e2 ,e1)
        (e2 e1)])
```

以上のように `lambda` を用いることで，パターンマッチによって `(e2 e1)` とされた際に，
`((lambda (x) (+ x 1)) 2)` の処理が行われる．ここでの局所変数 `x` が束縛変数としてはたらき，`lambda` の機能によって `(+ x 2)` へ 2 の代入が行われ，結果 3 が返される．そのため，先行研究中の `subst` のような代入を行う補助関数を定義せずに λ 計算が直感的に表現できる．

本研究では λ 計算に限らず，任意の束縛を持つ構文を定義可能な機能を HOAS を用いて実現することを目標とする．

3 設計

本節では，提案する LwRed の拡張において，任意の束縛を含む構文を定義するための機能 `define-term` について説明する．ユーザは，`define-term` というマクロ呼び出しによって構文を定義することができる．この呼び出しによって

- 構文名
- 束縛変数
- 参照する束縛変数

を任意で決定する．束縛変数や参照する束縛変数は，指定するためのキーワードとして，それぞれ `bind` と `in-scope-of` を用いる．また，`match` を利用することで，`define-term` によって定義した構文と合致する形の式を実行することができる．以下に `my-let` という束縛の起こる構文を定義した場合の実行例と結果を示す:

```
(define-term (my-let ([(bind z) e])
                     (in-scope-of body (z))))

(match (my-let ([a 1]) (+ a 2))
   [(my-let ([y f]) b #:subst sb)
    (sb y f b)])
```

define-term によって構文の名前や形状を指定し，それに従う構文を match 以降で呼び出す出すことで処理が行われる．define-term によって指定された形状以外で以降の呼び出しを行うとエラーが表示される．また，キーワード#:subst を用いて現在 sb としている代入操作の関数名を自由に設定することができる．

4 実装

本節では define-term がどのように定義されているかについて説明する．また，define-term 中で用いられる補助関数や機能についても説明する．

define-term は define-syntax によって定義され，主に補助関数を参照する with-syntax 部，ユーザが定義したい構文 name-struct を定義し，定義した項に対する match 式で用いられるパターンを適切な形状に展開する define-match-expander 部で構成される．define-syntax による定義とパターンマッチ，with-syntax による補助関数の参照は以下のように行われる:

```
(define-syntax (define-term stx)
  (syntax-case stx ()
    [(_ (stx-name body ...))
     (with-syntax ([(b ...)
         (find-binds  #'(body ...))]
         .... ) ....
```

define-term 構文は (_ (stx-name body ...)) という形式でパターンマッチが起こり，with-syntax 部によって補助関数である find-bind を body ... 部に適用したものを (b ...) という表記で用いることができる．find-bind は define-term 内で bind に指定されている識別子を抽出するための補助関数であり，その他必要になる補助関数を同様の方法で参照している．define-match-expander 部は以下のとおり:

```
(define-match-expander stx-name
 (λ (stx)
  (syntax-case stx (cons quote)
   [(_ body* ... #:subst sb)
    ....
    #'(app (λ (x) (list subst
                x 'dummy))
           (list sb
           (name-struct in ... rase ...)
           b ...)))])))
```

define-match-expander では，
(_ body* ... #:subst sb) という形状で match 式が書かれた場合，
(list sb (name-struct in ... rase ...) b ...)) へマッチするように展開されている．b ... と同様に in ... や rase ... も補助関数を適用したものの呼び出しであり，name-struct は define-term 内で構造体 (name-struct f rase ...) として定義され，HOAS を用いた
(name-struct (lambda (b ...) in ...) rase ...) という形で利用される． また，#:subst によって指定された sb は match 式を展開する際に関数 subst と対応させられる．関数 subst は define-term 内で定義される関数であり，構造体に格納されたそれぞれを，((lambda (b ...) in ...) rase ...) として，Racket の機能である lambda の処理が行われる形に変換する．

5 まとめと課題

本研究では，遷移系記述言語 LwRed において，束縛を表現可能な構文を定義する機能を実現するため，define-term の設計を行った．define-term は HOAS という手法を用いて，LwRed のホスト言語である Racket の機能 lambda を利用することで実装を行った．現状では，単純な λ や Racket における let のような束縛を持つ構文を処理することは可能であるが，letrec や let*のような複雑なスコープの構文を実装することは不可能であるため，束縛を完全に表現することが出来ない．よって，完全に束縛を表現するため，補助関数や define-term 本体の動作の見直しを行うことが今後の課題であると考える．

参考文献

[1] Seiji Umatani, *Lightweight DSL for Describing Extensible Transition Systems.* https://dl.acm.org/doi/10.1145/3605098.3636025.

[2] Frank Pfenning, Cod Elliott, *Higher-Order Abstruct Syntax.* Department of Computer Science Carnegie Mellon University Pittsburgh, Pennsylvania 15213-3890, 199-208, 1988.

[3] *Racket Guide.* https://docs.racket-lang.org/guide/index.html.