神奈川大学理学部　情報科学科　馬谷研究室

Scheme サブセットの操作的意味論の Redex による実装

武田 雄一郎 201803709

1　はじめに

抽象解釈器の構築において Scheme 言語派生の Racket 言語のライブラリである Redex を用いる．Redex とは文法と簡約関係を記述し定義する事で実行可能な仕様を作成するライブラリである．

Redex を用いる利点は，`traces` 機能を用いることで簡約の各ステップを簡約グラフを使って網羅的に可視化できることである．

最終的には Scheme サブセット [3] を基にした抽象解釈器の構築を Scheme サブセットの Redex による実装，継続データのヒープ化 [4]，ヒープ化したデータ構造の有限化 [5] という手順により行う．本研究ではこれら手順の一つ目である Scheme サブセットの Redex による実装を行う．

2　背景

抽象解釈とはプログラムを抽象化によって単純化し近似的に実行してプログラムの解析を行うことである．プログラムを近似的に実行することで不必要な部分を無視し必要な結果だけを部分的に求めることができるため効率的にプログラムの解析を行うことができる．

例えば，符号解析の場合 3 と -100 という数値を符号に抽象化した場合その符号は 3 は $(+)$，-100 は $(-)$ となる．

また，数値で計算をしてからその符号の結果を確認する通常の解釈とは違い抽象解釈は符号だけで計算する．$3+(-100)=(-97)$ を抽象解釈を用いて計算すると $(+)+(-)=(+/0/-)$ となる．ここで，$(+)$ は正の抽象値，$(-)$ は負の抽象値，$(+/0/-)$ は正，ゼロ，負のどの場合もあり得る抽象値を表す．

3　文法と簡約定義

本章では Scheme サブセット [3] の Redex による実装を行う上で Redex の使用方法について説明する．

まず，Scheme サブセットの文法を正確に定義するために `define-language` 形式で文法を使用する．

```
(define-language SCM
(p (store (sf ...) e))
(sf (x v))
(e (e e ...)
  (set! x e)
  (begin e e ...)
  (if e e e)
  x
  v)
(v (lambda (x ...) e)
    n  #t #f - unspecified)
      /*省略*/
```

`p` は変数 `x` から値 `v` をマッピングする `store` と式 `e` で構成され，プログラムの実行状態を表す．`sf` は `store` 中のフィールドで，変数 `x` に値 `v` が格納されていることを表す．`e` は関数呼び出し，`set!`式，`begin` 式，`if` 式，変数 `x`，値 `v` を表す非終端記号である．`v` はラムダ式，数値 `n`，ブール値，否定演算子，`unspecified` という値を表す．

簡約関係は `reduction-relation` を用いて定義する．簡約関係の形式は言語名，与えられた領域，`-->`で示されたいくつかの節で定義され，左辺，右辺， 任意の数の `side condition`，任意の名前からなり一連の書き換え規則を受け取り二項関係を定

義する．

SCM の簡約関係 rS を以下のように定義する．全部で 7 つある規則のうち，ここでは 2 つだけを示す．

```
(define rS
(reduction-relation
SCM #:domain p
(-->
  (store (sf_1 ... (x_1 v_1) sf_2 ...)
         (in-hole E (set! x_1 v_2)))
  (store (sf_1 ... (x_1 v_2) sf_2 ...)
         (in-hole E  100))
  "Mset")
(-->
  (store (sf_1 ... (x_1 v_1) sf_2 ...)
         (in-hole E x_1))
  (store (sf_1 ... (x_1 v_1) sf_2 ...)
         (in-hole E v_1))
  "Mlookup")))
```

P，E は，それぞれ p，e 中の簡約可能箇所が穴になっている文脈である．

1 つ目の Mset 規則とは関連する値をストア内の指定された識別子を指定された置換文字列で置き換える規則である．2 つ目の MLookup 規則とは，変数の評価に対応しており，変数を store 内の関連する値に置き換える規則である．

このように Redex を用いるには文法と簡約関係を定義する必要がある．

4　実行例

```
(store ((x 1) (y 3) (z 5))
 ((set! x y) (set! x z)))
```

を 3 章で定義した文法，簡約関係から traces 機能を用いて以下のコードを実行する．

```
> (traces rS
    (term (store ((x 1) (y 3) (z 5))
     ((set! x y) (set! x z)))))
```

実行した結果，図 1 のように簡約グラフが表示される．図 1 の簡約グラフから SCM の簡約関係を正しく定義できていることが確認できる．

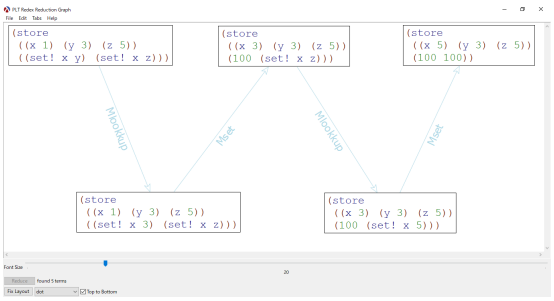

図 1　Redex による簡約グラフの可視化

5　まとめ

本研究では Scheme サブセットの Redex による実装を行った．Scheme サブセットを基にした抽象解釈器の構築において Redex による実装が本研究により可能であることを確認した．

参考文献

[1] Robert Bruce Findler，Casey Klein，Burke Fetscher，Matthias Felleisen，Redex: Practical Semantics Engineering．https://docs.racket-lang.org/redex/index.html

[2] David Van Horn，An Introduction to Redex with Abstracting Abstract Machines（v0.6）．https://dvanhorn.github.io/redex-aam-tutorial/

[3] Jacob Matthews，Robert Bruce Findler，An Operational Semantics for Scheme，Journal of Functional Programming，18(1)，47–86，2007.

[4] 山本怜，Redex 上の Scheme インタプリタにおける抽象解釈のための継続のヒープ化，2021 年度神奈川大学理学部情報科学科卒研要旨集，2022.

[5] 森田翔大，Redex 上の Scheme インタプリタにおけるヒープの有限化による抽象解釈の実現，2021 年度神奈川大学理学部情報科学科卒研要旨集，2022.