B-014

VDM-SLの陽仕様記述からLispファミリ言語Schemeへの変換

Conversion from Explicit Specifications in VDM-SL to Scheme as Family of Lisp Language

長谷 卓容 †　　和崎 克己 †
Takahiro Hase　Katsumi Wasaki

1　はじめに

Floyd-Hoare 論理において，あるプログラムについて，部分的正当性の表明を導出することを考えるとき，そのプログラムが満たすべき事前条件・事後条件を，それぞれそのプログラムの仕様の一種とみなす．その仕様を記述する言語として仕様記述言語 VDM-SL[1, 2, 3, 4] が存在する．VDM-SL は設計上の仕様であるため，そのままの状態では実行可能ではなく，かつシミュレーションを目的としたとき，外界に相当する部分は手作業で準備する必要があった．このため VDM-SL 上での記述と，それに対応する実装で仕様通り動作することの確認とを，一貫して検証する環境が必要と考えた．本稿は仕様記述言語 VDM-SL から関数型言語 Scheme への変換とコード生成器に関する検討を行ったものである．

はじめに仕様記述言語 VDM-SL の概要，そして仕様記述の陽仕様，陰仕様についてそれぞれ “挿入ソート”，“最大公約数”をもとめる仕様記述も用いて述べる．

また，本研究では Lisp ファミリ言語 Scheme[5] への変換を考えている．本研究で接続先とし て考えている Scheme とは，Lisp の方言の 1 つで，関数型プログラミング言語としてよく知られており，前置記法，型の観念がない，lambda 記法などの特徴がある．Lisp ファミリ言語 Scheme について紹介したのち，VDM-SL で記述可能なこと，Scheme で記述可能 なことなどの差異をはっきりさせるために，具体的には “挿入ソート ” と “マージソート ” を VDM-SL から Scheme へ機械的に変換した場合について確認を行った．最後に VDM-SL 上で記述された陽仕様を Lisp ファミリ言語 Scheme 上の関数へ変換する際の，種々の方策について述べ，変換する上での問題点について明らかにする．

2　仕様記述言語 VDM-SL

2.1　VDM-SL 概要

通常，コンピュータに与えられる命令はプログラムとして記述されている．プログラムの正しさについて論ずるために，プログラムが満たすべき事前条件，事後条件をそれぞれプログラムの仕様の一種と見なし，プログラムの正当性を定義する．このプログラムの正当性を証明するために, 新たな公理と推論規則を追加し, 既存の論理体系を拡張した Floyd-Hoare 論理がある．Floyd-Hoare 論理において，あるプログラムにおいて，部分的正当性の表明を導出することを考えるとき，そのプログラムが満たすべき事前条件・事後条件をプログラムの仕様の一種とみなす．その際，仕様そのものを陽に記述するための道具として VDM-SL（Vienna Development Method Specification Language）と呼ばれる仕様記述言語がある．

VDM-SL の特徴として，ライトウェイトな手法であるため，数学やソフトウェア工学の特殊な知識はあまり要求されず実際の開発現場に導入しやすいこと．また、無料で使用できる開発支援ツール（VDMTools）[6] があること，事前，事後，不変条件を記述する構文があること，海外の開発現場での採用実績が数多くあることがあげられる．VDMTools はデンマークの Peter Gorm Larsen 博士が開発した VDM 開発支援ツー ルであり，形式技術と実用技術を巧みに統合している．現在は仕様の構文，型チェック機能， 証明課題生成機能， 実行可能仕様のインタープリタとデバッグ機能，実行可能仕様のコードカバレージ計測機能などの機能が提供されている．

2.2　陽仕様

VDM-SL には陽仕様と陰仕様による 2 通りの仕様記述が存在する．陽仕様は値を直接的に示すのに対して，陰な表現は満たすべき性質を示している．以下に陽仕様である陽関数，陰仕様である陰関数の各々の例を挙げる．

2.2.1　陽関数

図 1 は挿入ソートの記述である [6]．関数 Dosort は与えられた数列を再帰的に分割し，数値と昇順にソートされている数列 sorted とを InsertSorted に送っている．関数 InsertSorted は与えられた数列へ与えられた数値が昇順となる位置への挿入を行っている．このように陽関数では値や手続きを直接示している．

2.2.2　陰関数

図 2 は最大公約数を求める仕様記述である [1]．補助関数 Cds は公約数を求める関数であり，2 つの引数 x, y の公約数を返すように，補助関数 Max は与えられた集合 ns の中の最大値を返すように post 以降に記述されている事後条件により定義されている．関数 gcd は 2 つの整数 x，y を引数にとり，関数を実行した結果が z = Max(Cds(x，y)) となる，2 つの引数の公約数の中で最大となるような z を返す関数である．このように陰関数は

† 信州大学大学院工学系研究科，Graduate School of Science and Technology， Shinshu University.

```
function
  DoSort: seq of real -> seq of real
  DoSort(l) ==
    if l = [] then
      []
    else
      let sorted = DoSort (tl l) in
        InsertSorted (hd l,  sorted);
  InsertSorted: PosReal * seq of PosReal
                                    -> seq of PosReal
  InsertSorted(i, l) ==
    cases true :
      (l = []) -> [i],
      (i <= hd l) -> [i] ^ l,
      others -> [hd l] ^ InsertSorted(i, tl l)
    end
```

図 1: VDM-SL による挿入ソートの記述 [6]

```
functions
    gcd(x:nat1,  y:nat1) z:nat1
    post z = Max(Cds(x, y));
    Cds(x:nat1, y:nat1) nset:set of nat1
    post forall z in set nset &
              exists n:nat1 & x=n*z and
              exists m:nat1 & y=m*z;
  Max(ns:set of nat1) mx:nat1
  post mx in set ns and forall n in set ns & mx>=n
```

図 2: VDM-SL による最大公約数を求める仕様記述 [1]

関数の値を求めるための手続きは記されておらず，満たすべき条件が記されている．

以下、本稿では陽仕様の中でも陽関数のみ取り扱う．

3 Lisp ファミリ言語 Scheme

Scheme[6] とは Lisp 風の関数型プログラミング言語で，最も言語仕様が小さい言語の一つである．

```
(* 5 9)
> 45
```

Scheme において，式の並びをかっこで囲んで手続きの作用を表現する 1 行目のような式を組合せという．また，並びの左の字の要素を演算子，他の要素を被演算子という．組み合わせの値は演算子が指定する手続きを，被演算子の値である引数に作用させて得る．演算子を左におく書き方を前置記法という．2 行目の > 45 は解釈系の応答で式の評価結果である．

```
(define (minus5 x) (- x 5))
  (minus5 23)
> 18
```

また，手続き定義の一般型は上記のように

```
(define (<name> <formal parameters>) <body>)
```

という形で表される．名前 <name> はこの手続き定義に対応づける名前である．仮パラメタ <formal parameters> は手続き本体の中で，手続きの対応する引数を指すための名前である．本体 <body> は仮パラメタが手続きを作用させる実引数で取り替えられて手続き作用の値を計算する式である．<name> と <formal parameters> は，かっこの中にまとめられ，この手続きを実際に呼び出す時と同じ形をしている．

また Scheme は手続きを引数としてとるか，あるいは手続きを戻り値とするような高階手続きを定義できる．ある手続きを高階手続きの引数として用いる場合，define により手続きを定義せずに，手続きを直接指定する方法として，lambda が存在する．lambda は手続きを作り出す特殊形式である．一般的に lambda は手続きに名前がつかない他は，define と同様に手続きを作り出すときに用いられ

```
(lambda (<formal parameters>) <body>)
```

により手続きは，環境で名前と対応づけられていないこと以外は define で定義されたものと同じ手続きである．

4 陽仕様の VDM-SL 記述から Scheme への変換

4.1 変換の手順

主に陽関数は

```
<identifier>':'<discretionary type> '->'<type>
<identifier> <parameters list> '=='
<function body>
[ <pre expression> ]
[ <post expression> ]
```

で定義される．<identifier>は関数の名前，<discretionary type>は引数の型，<type>は返り値の型，<parameters list>は変数名，<function body>は関数本体，<pre expression>は事前条件，<post expression>は事後条件を表している．{ } で囲まれた構文は 0 回以上出現する構文項目，[ ] で囲まれた構文はオプションの構文項目を意味している．

また <discretionary type> <parameters list>の並びは対応している．Scheme は先述の通り

```
(define (<name> <formal parameters>) <body>)
```

で表される．VDM-SL の定義と Scheme の定義を対応させると

```
<identifier> -> <name>
<parameters list> -> <formal parameters>
<function body> -> <body>
```

となり

```
(define (<identifier> <parameters list>)
                                    <function body>)
```

となる．Scheme の特徴として型，事前条件，事後条件の観念がないことが挙げられる．そのため VDM-SL から Scheme へ変換する際<discretionary type>，<type>，<pre expression>，<post expression>については考慮する

```
(define (DoSort seq)
 (if (null? seq) null
    (InsertSorted (car seq)
                  (DoSort (cdr seq)))))
(define (InsertSorted num seq)
 (cond ((null? seq) (cons num null))
       ((<= num (car seq)) (cons num seq))
       (else (cons (car seq)
                   (InsertSorted num (cdr seq))))))
```

図 3: Scheme による挿入ソートの記述

```
functions
  MergeSort: seq of real -> seq of real
  MergeSort(l) ==
    cases l:
      [] -> l,
      [e] -> l,
      others -> let l1^l2 in set l be st
       abs (len l1 - len l2) < 2  in
                 let l_l = MergeSort(l1),
                  l_r = MergeSort(l2) in
                Merge(l_l,  l_r)
  end;
Merge: seq of int * seq of int -> seq of int
Merge(l1, l2) ==
  cases mk (l1, l2):
    mk ([], l), mk (l, []) -> l,
    others -> if hd l1 <= hd l2 then
               [hd l1] ^ Merge(tl l1,  l2)
              else
               [hd l2] ^ Merge(l1, tl l2)
  end
pre forall i in set inds l1 & l1(i) >= 0 and
     forall i in set inds l2 & l2(i) >= 0
```

図 4: VDM-SL によるマージソートの記述 [6]

```
(define (MergeSort seq)
  (cond ((null? seq) seq)
        ((element? seq) seq)
        (else (let ((l1 (Sep1 seq (len seq)))
                    (l2 (Sep2 seq (+ 1 (len seq)))))
                (let ((l_l (MergeSort l1))
                      (l_r (MergeSort l2)))
                  (Merge l_l l_r))))))
(define (Merge seq1 seq2)
  (cond ((null? seq1) seq2)
        ((null? seq2) seq1)
        (else
         (cond ((<= (car seq1) (car seq2))
                (cons (car seq1)
                      (Merge (cdr seq1) seq2)))
               (else
                (cons (car seq2)
                      (Merge seq1 (cdr seq2))))))))
(define (element? seq)
  (cond ((null? seq) #f)
        ((not (null? (cdr seq))) #f)
        (else #t)))
(define (len seq)
  (quotient (length seq) 2))
(define (length items)
  (if (null? items)
      0
      (+ 1 (length (cdr items)))))
(define (Sep1 seq len)
  (if (= len 0)
      ()
      (cons (car seq) (Sep1 (cdr seq) (- len 1)))))
(define (Sep2 seq len)
  (cond ((= len 1) seq)
        (else (Sep2 (cdr seq) (- len 1)))))
```

図 5: Scheme によるマージソートの記述 1

必要はない．関数本体部分の対応については以下に”挿入ソート”，”マージソート”の例を用いて説明する．

4.2 変換例”挿入ソート”

図 3 は図 1 に示した VDM-SL による挿入ソートの記述を Scheme へ書き換えたものの一部である．VDM-SL による挿入ソートの記述に陰的な表現はなく Scheme への変換は一意に決まるため，VDM-SL における case 文，hd，tl，[ ]，^をそれぞれ，Scheme の cond 文，car，cdr，null，cons に，また関数定義，let 文の記述を変更した．これは一意に変換することができた好例である．

4.3 変換例”マージソート”

図 4 はマージソートの記述である．また事前条件として図 16 の 23， 24 行目において，l1 のインデックスを i としたとき l1(i) は 0 以上であり，また l2 においても同様の条件を述べている．図 4 のマージソートの記述ではリストの分割については陰な表現が用いられており，リストの分割方法を一意に決定することができない．そのために列の分割方法を検討する必要がある．リストを半分に分割するためにはリストの長さが分かっていなければならない．そのため今回はリストとリストの半分の長さを調べ，前半部分を求めるためにはその長さの分だけ前から切り出し，後半部分を求めるためには，リストの残りの列を求めることによりリストを分割したものが図 5 の Scheme である．

Scheme には基本手続きとして，与えられたリストが空でない単位列であるかを判断する関数，リストの長さを調べる関数，指定した場所で列を分割する関数が備わっていない．そのため単位列であるかを調べる関数 element?，リストの長さを調べる関数 length，リストの半分の長さを求める関数 len，指定した場所で列を分割する関数 Sep1，Sep2 を新たに定義した．分割する関数が 2 つ定義されているのは列の前半部分と後半部分が必要となるためである．しかし，図 4 の 7，8 行目の

```
let l1^l2 in set {l} be st abs (len l1 - len l2) < 2
```

はリスト l を l1，l2 に分割する際，“l1，l2 のリストの長さの差の絶対値が 2 未満になるように分割する”というリストの分割方法について示している．ここで，もし分割の対象にあるリストの長さが偶数（2m）の場合，図 6 のようにリストの分割方法は前半部分のリストの長さ

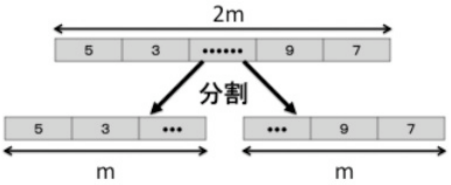

図 6: 偶数の場合の分割方法

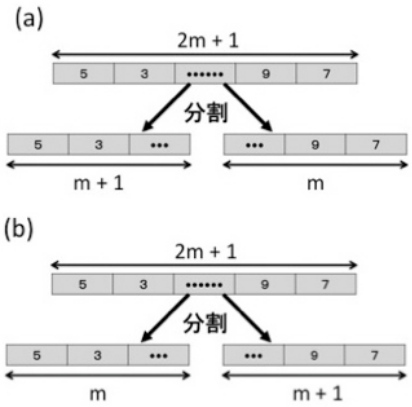

図 7: 奇数の場合の分割方法

```
(define (MergeSort seq)
 (let ((ran (random 1))
       (leng (length seq)))
   (cond ((null? seq) seq)
     ((element? seq) seq)
     (else (cond ((= 0 (remainder leng 2))
                  (let ((l1 (Sep1 seq (len seq)))
                        (l2 (Sep2 seq (+ 1
                                         (len seq)))))
                    (let ((l_l (MergeSort l1))
                          (l_r (MergeSort l2)))
                      (Merge l_l l_r))))
                 ((and (= ran 0) (< 3 leng))
                  (let ((l1 (Sep1 seq (- (len seq)
                                         1)))
                        (l2 (Sep2 seq (len seq))))
                    (let ((l_l (MergeSort l1))
                          (l_r (MergeSort l2)))
                      (Merge l_l l_r))))
                 (else (let ((l1 (Sep1 seq
                                       (len seq)))
                             (l2 (Sep2 seq
                                       (+ 1
                                         (len seq)))))
                         (let ((l_l (MergeSort l1))
                               (l_r (MergeSort l2)))
                           (Merge l_l l_r)))))))))
```

図 8: Scheme によるマージソートの記述 2

m，後半部分のリストの長さ m と一意に決定することができる．しかし，リストの長さが奇数（2m+1）の場合，前半部分のリストの長さが m+1 で後半部分のリストの長さが m で分割するのか，もしくは前半部分のリストの長さが m で後半部分のリストの長さが m+1 で分割するのかは図 4 の VDM-SL による記述にはない．よって，図 7 のようにリストの長さを一意に決定することができない．そのため新しく分割方法をランダムに決定するための局所変数 ran を定義する．図 8 にリストの長さが奇数の場合，ran の数値によってランダムに前半部分，後半部分の長さを決定するように修正したものを示す．

VDM-SL から Scheme への変換は，VDM-SL に陰な記述がされていなければ忠実に Scheme へと変換することができるが，図 8 の例の場合，陰な記述があれば VDM-SL から Scheme へ変換する際．意味的な解釈を加えた上で，テンプレート等に基づく変換を行う必要がある。

5 まとめと今後の課題

VDM-SL の陽仕様記述から Scheme への変換について，まとめと課題を以下に挙げる．

（1）基本手続き: VDM-SL から Scheme への変換は，VDM-SL に陰 な記述がされていなければほぼ忠実に Scheme へと変換 することができるが，VDM-SL には存在し Scheme には存在していない基本手続きがある．その基本手続きについて Scheme にて記述し，テンプレート化しなければならない．

（2）陰な仕様の検討: VDM-SL には Scheme へ一意に変換できないような，陰な表現の記述 (例:マージソートの記述) が存在する．そのような記述に対してのアプローチも考えなければならない．

（3）他の定義達について: また，VDM-SL には関数定義以外にも多数の定義が存在している．本稿において記載されている VDM-SL は陽関数定義だけであるので，他の定義達についても検討しなければならない．

（4）変換システムの構築: 今回変換はすべて手動で行ったが， VDM-SL から Scheme へと自動で変換できるような変換システムの構築．

（5）VDM-SL と Scheme の動作の同等性: 図の関数 random のように実際に駆動することができないため，果たして VDM-SL で書かれた記述と Scheme での記述が一致しているか確認できない場合がある．

参考文献

[1] 荒木啓二郎, 張漢明: ‘‘プログラム仕様記述論”, オーム社 (2002-11)

[2] 佐原伸: ” 形式手法の技術講座”, 株式会社ソフト・リサーチ・センター (2008-4)

[3] J.Fitzgerald, P.G.Larsen: “MODELLING SYSTEMS”, Cambridge University Press (1998)

[4] VDM ポータルサイト: http://www.vdmportal.org/twiki/bin/view

[5] G.J.Sussman, J.Sussman, H.Abelson: ”Structure and Interpretation of Computer Programs”, The Massachusetts Institute of Technology (1998)

[6] VDMTools information web site: http://www.vdmtools.jp/