集合および関係演算に基づくプログラミング言語Rの支援環境

2D-2

木村英一・内田智史・間野浩太郎

(青山学院大学 理工学部 経営工学科)

1. はじめに

我々は、集合および関係演算に基づく汎用プログラミング言語Rを設計・開発中である。この言語は、通常の手続き型言語とは異なり、扱うデータが集合・関係を基に記述されている。従って、常用言語とは全く異なる支援環境が必要となる。まず、集合・関係を基にしてデータを表現するには、現在あるようなASCII(JIS)コードによる文字表現だけではなく、数学での記号(Σ etc.)や漸化式などで見られる添字(Xi etc.)などで表現する方が優れている。また、プログラム上において手順、すなわちタイミングを考えることが少なくなり、むしろ、集合として記述されたデータ構造の正しさを検証することが重要となる。また、手順中心のデバッッキング手法も再考が必要となる。例えば、シンボリックデバッガの構成も他の言語とは異なる形式となる。つまり、ブレイクポイントやトレースという概念はない。本報告では、これらの点を踏まえて、現在設計，開発中の支援環境について述べる。

2. システム構成

本支援システムの構成は、

1.専用エディタ
2.インタプリタ＋デバッガ

である。

2.1.専用エディタ

R言語では、常用言語のようなASCII(JIS)コード文字，漢字だけでなく、数学での記号や添字(Σ,Xiなど)をプログラム中に記述できる。さらに、フォントの違いで、同じ文字列を区別する(例えば、予約語のフォントと、通常のテキストのフォントとを区別するなど)。そこで、これらの記号の扱いや、フォントによる区別が可能であるビットマップ方式のエディタが必要である。

2.2.デバッガ

R言語でのプログラムにおける、手続きの流れは、ソースプログラムレベルではほとんど存在しない。そのため、ブレークポイントやトレースといった概念は存在せず、現在のシンボリックデバッガの構成とは異なった形式のデバッグが必要である。

例えば、手続き型言語では、制御構造の入口や出口付近に不正要因となりうるバグが存在する可能性が高い。例えば、ループ回数やループ脱出条件である。これに対して、R言語でのそういったエラーは、データ構造の定義に暗黙に制御構造が含まれているため、プログラマにはその制御構造は直接見えず、ほとんどない。そのデータ構造定義上に多くの不正要因が存在するようになると思われる。つまり、データ構造の定義に関するエラーを取り除くことが、デバッグ作業の主となると考えられる。

そこで、本デバッガでは、視覚的にデータ構造を表現し、プログラマの問い合せにより、データが加工される際の適用規則を表示するといったより分りやすいものを考えている。つまり、入力データ構造表示部では、入力データ構造を定義した形式で表示する。出力データについても同様である。データ加工中間表示部では、ある規則によって加工されたデータを表示する。また、データが加工された際、どの規則が適用されたかを表示する(図 1)。

3. 実例

ここで、簡易Basicインタプリタを例に、デバッグについて説明する。プログラム例 1が簡易Basicインタプリタのプログラムである。例えば、

```
10 J=0
20 FOR I=1 TO 10
30    J=J+I
40 NEXT I
50 PRINT J
```

をこのインタプリタにかけ、それをデバッグすることを考える。

Line は文番号とStatementと改行とで構成されている。そこで、入力データ構造表示部にこの Line で定義された構造で読み込んできたデータを表示する。つまり。最初は "10 J=0" を表示する。そして、インタプリタではこの文を解釈する。それは、Rule 部に記述された規則に従って解釈するので、ここではRule部の LET:[LeftVar... が適用される。その結果、データの加工が行われたものが加工中間表示部に表示される。この場合では、 VName = J,VValue=0,そして、VariableTable に登録されたことを表示する(図 2)。また、ここでプログラマから、どの規則が適用されたかの問い合せがあった場合、その加工で適用した規則、ここでは、リスト1の50-51行目の "LET:[LeftVar:VarSet "=" ..." を表示する(図 3)。文番号10ではまだ、出力データがないので、出力データ構造表示部には何も表示されない。次に、文番号20を同様にして解釈実行する。なお、この場合は文番号20-40が FOR-BLOCK という一つのまとまりである(Rule部の FOR-BLOCK:の項)から、この 3行を 1行とみなして処理する。最後に、文番号50を入力データ構造表示部に表示し、出力データ構造表示部に、(この場合はディスプレイに)出力するオブジェクトを、そのデータ構造の形式で表示する。つまり、I の値を表示する(図 4)。

Supporting Environment for A Programing Languege R based on Set and Reletion Opretion

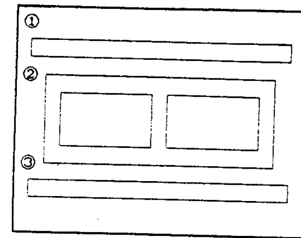
Eiichi Kimura,Satoshi Uchida,Koutaro Mano

Aoyama Gakuin University

```
BasicInterpreter:Program
    { Arg[FileNameStr:string]
      Def{ProgramFile:SequentialFile(FileNameStr)
             [*Line[文番号:numstring Statement.'Yn']]:
      Statement: ↓(Separator)
             {REM IF FOR-BLOCK PRINT INPUT GOTO LET}:
      Separator{' ' 'Yn' 'Yt' nil}:
      ThemBlock:{*Statement}:
      ElseBlock:{*Statement}:
      VariableTab:DynamicSet{*Variable[VName:key(string).VValue:int]}
      Alpha:char{'A' 'B' 'C' ... 'Z'}:
      Num:char{'0' '1' '2' ... '9'}:
      AlphaNum{Alpha Num}:
      VarSet:string[Alpha <0>*AlphaNum]:
      Var:string[Alpha <0>*Alpha]=>VValue:[VName=Var]:
      Constant:string[<{'+' '-'}> a:[<1>*Num] <'.' b:[<0>*Num]#0
                   <'E' <{'+' '-'}> c:[<0>*Num]#0>>]
                                            =>number([a  b c]):
      VarC:{Var Constant}
      LogicalOperation:string{"=" "<>" ">" "<" "<=" ">="}
      Lable:[<5>Num]
      符号:{'+' '-' nil}:
      基本演算子:{'+' '-'}:
      拡張演算子:{'*' '/'}:
      expression:[符号 項 *[基本演算子 項]]:
      項:[因子 [拡張演算子 因子]]:
      因子:{Constant Var : ↓(nil)['(' expression ')']}:
      FOR:["FOR" CtrlVar:VarSet "=" Init:VarC "TO" Final:VarC
                                         <"STEP" Step:VarC>]:

Rule{ REM:["REM" *:char]=>nil:
      FOR-BLOCK: ↓('Yn')[FOR *line NEXT]
       =>[
          [VName=CtrlVar.VValue=init]→VariableTab:
           :repeart[
            eval([Statement!all]):
            if NextVar ≠ CtrlVar then NextMissingError():
            [Vname=CtrlVar.VValue=VValue:[VName=CtrlVar]+Step]→VariableTab:
             Value:[VName=CtrlVar] > Final
            ]
         ]
     IF:["IF" ifexpression "THEN" ThenBlock <"ELSE" ElseBlock>]
             =>if eval(ifexpression) = t then eval(ThenBlock)
                                      else eval(ElseBlock):
     PRINT:["PRINT": ↓(',')[*print-expression:Var]]
              =>[print-expression!all]→Display:
     print-expression:[printVar:Var]=>VValue:[VName=printVar]:
     INPUT:["INPUT": ↓(',')[*input-expression:Var]]
              =>KeyBoard→[input-expression!all]:
     input-expression:[inputVar:Var]=>(VName=inputVar.VValue=(—)]
                                                          →VariableTab:
     GOTO:["GOTO" label]=>GOTOProc(lable):
     LET:[LeftVar:VarSet "=" expresion]
             =>[VName=LeftVar.VValue=eval(expression)]→VariableTab:
     ifexpression:[Var1:Var LogicalOperation Var2:Var]
             =>[:case[LogicalOperation="=" => Var1 = Var2:
                     LogicalOperation="<>" => Var1 <> Var2:
               ]
 :
Proc[Interpreter:eval([Statement!all(i)]):]
GOTOProc(label)[ifexist j!文番号j = label then
                           [i←j: reeval(Interpreter)]
                       else
                           GOTOstatementError(label):]
GOTOstatementError(label)[Display("このラベルは宣言されていない。"):]
NextMissingError()[Display("for文に対するnextが一致していない。"):]
```

プログラム例1　簡易BASICインタプリタ



① 入力データ構造表示部
② データ加工中間表示部
③ 出力データ構造表示部

図1　デバッガの画面

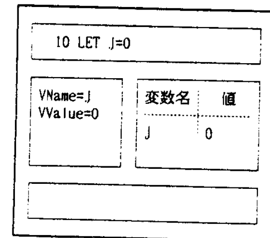


図2　デバッグの実際 1

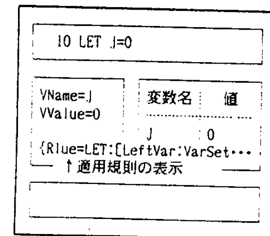


図3　デバッグの実際 2

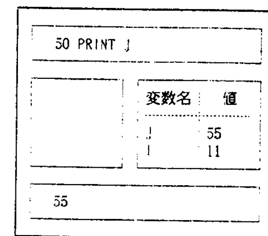


図4　デバッグの実際 3

4．まとめ

今回は、主にデバッキングシステムについて述べた。このデバッガは手続き型言語のものとは異なる点が多く、今後、さらにこれらの相異点や新しいデバッグ手法など検討する点が多い。

[参考文献]

1.内田智史・木村英一・間野浩太郎、他：オブジェクト指向に基づくアセンブリ言語プログラミング環境　情報処理学会第32回全国大会 6F-7,pp.453-454(1986.3)

2.内田智史・木村英一・間野浩太郎：集合および関係演算に基づくプログラミング言語Rの基本機構について、情報処理学会第33回全国大会 2D-1,pp.373-374 (1986.10)