4G−4

表示的意味記述からのプログラミング環境の生成

新井正樹　武田正之　井上謙滅

東京理科大学

1　はじめに

対象とするプログラミング言語の表示的意味論に基づくプログラミング環境仕様記述から対話的なプログラミング環境を生成する方法を提案する。プログラミング環境仕様記述は対象言語の表示的意味記述に実行・トレース・デバッグ対象の指定、ウィンドウシステム上での各種情報の表示方法、プログラムの構造表示のための指定などを追加して拡張した表示的意味記述（以下EDSDと略す）である。EDSDから生成されたプログラミング環境は、ユーザのプログラムを各種の情報を埋め込んだ属性付きラムダ式に変換する。プログラミング環境の各種の機能はこの属性付きラムダ式の簡約時の副作用によって実現される。

2　システムの構成

対話型プログラミング環境生成系は図1に示すようにEDSDコンパイラとプログラミング環境カーネルで構成されている。システム管理者はEDSDコンパイラを用いて対象言語のEDSDから字句解析系、構文解析系、表示的意味生成系、プログラミング環境で使用するユーザインタフェースを生成し、それらをプログラミング環境カーネルと結合して対象言語のプログラミング環境を生成する。プログラミング環境カーネルは対象言語に依存しない部分である。

EDSDは意味領域定義部、意味関数定義部、非終端記号宣言部、終端記号宣言部、補助関数定義部、初期状態定義部、構文規則意味定義部、字句パターン定義部、終端記号意味定義部からなる。EDSDの例を図2に示す。

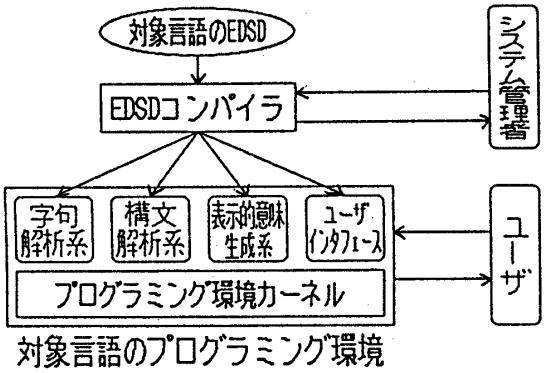

図1: 対話型プログラミング環境生成系

Generating Programming Environments
from Denotational Semantic Descriptions
Masaki ARAI, Masayuki TAKEDA, Kenzo INOUE
Science University of Tokyo

```
SD N (n) = integer {default};
SD Var (v) = string {default};
SD S (s) = Var -> N {default};

MF mc : S -> S;
MF me : S -> N;

Vn Program, Cmd : mc;
Vn Exp : me;

Vt 'id' : Var;

INIT = "function undef: DERROR";

Program --> Cmd () {"($1 $s)"}
Cmd --> Cmd ';' Cmd () {"($2 ($1 $s))"}
Cmd --> 'id' ':=' Exp (execute)
 {"(update $1 ($2 $s) $s)"}
Exp --> 'suc' Exp () {"(plus ($1 $s) 1)"}
Exp --> 'id' () {"($s $1)"}
Exp --> 'zero' () {"0"}

PAT upperLetters = "A|B|C|D|E|F|G";

'suc' --> "suc"
'zero' --> "0"
':=' --> "="
';' --> ";"
'' --> " |\t|\n"
'id' --> "$upperLetters" {"$pattern"}
```

図2: EDSDの例

3　属性付きラムダ式とその簡約手続き

定義1 属性付きラムダ式はラムダ抽象$(\lambda x.M)$に属性$A' \subseteq Att$が付いたもので、$(\lambda\{A'\}x.M)$で表す。ここで、Attはすべての属性の集合である。属性付きラムダ式は、部分項、束縛変数、自由変数、α変換の定義に関してラムダ抽象と同等である。

以下属性付きラムダ式もラムダ式とする。次に属性付きラムダ式簡約手続きを定義する。

定義2 属性付きラムダ式簡約手続き$ALTR$は、引数にラムダ式Mを取り、次のアルゴリズムで動作する手続きである。

```
procedure ALTR(M):
begin
   T := M;
   while not isNormalForm(T) do
      T := Δ(T);
   return T
end
```

ここで、$isNormalForm$は引数が正規形のラムダ式かどうか判定する手続きである。Δは内部状態をもち、引数がラムダ式の手続きで、ラムダ式を最左戦略で簡約し、その副作用で内部状態を変化させたり、他の手続きを実行したりする。Δの動作は、現在の内部状態、引数のラムダ式T、もしTの最左リデックスが$(\lambda\{A'\}x.L)K$の形をしている場合はその属性A'によって決まる。また、Δの初期状態は手続き$ALTR$が呼び出される前に設定される。

4　生成されたプログラミング環境の機能

EDSDから生成されたプログラミング環境はユーザのプログラムを字句解析系、構文解析系によって、位置情報付き構文解析木に変換する。この位置情報付き構文解析木の各節と葉はユーザのプログラムのどの部分に相当するかの位置情報を持つ。表示的意

味生成系は作成された位置情報付き構文解析木からユーザのプログラムの静的表示的意味と動的表示的意味をそれぞれ表すラムダ式を生成する。これらのラムダ式を簡約することが、それぞれ静的意味誤りがないかどうかの検査とプログラムの実行に相当する。

以下、例をあげて説明する。

生成規則 $X_0 \rightarrow X_1 \cdots X_n$ で表されるプログラムテキストを実行・トレース・デバッグの対象としたいときには EDSD で以下のように記述する。

$$X_0 \rightarrow X_1 \cdots X_n()\{M_s\}(execute)\{M_d\}$$

M_s と M_d はそれぞれこの生成規則の静的表示的意味記述と動的表示的意味記述である。ここで、X_0 の意味は $D_1 \rightarrow D_2 \cdots \rightarrow D_q$ の型を持つ意味関数で与えられると定義されているとする。またすべての属性の集合を $Att = \{execute, p\}$ とする。属性 *execute* は実行・トレース・デバッグの対象に付ける属性であり、p は表示的意味生成系が埋め込む位置情報である。このとき、この生成規則から導出されるプログラムテキストの動的表示的意味は以下の式で表される。

$$\lambda\$1 \cdots \$m.(\lambda\{execute, p\}\$d_1 \cdots \$d_{q-1}.M_d)$$

ここで、$\$d_i$ は意味領域 D_i 上を動くメタ変数の名前に\$記号を付けたものである。また$\$i$ は生成規則の右辺の、意味をもつ i 番目の文法記号の表示的意味を表す。右辺が空もしくは、右辺のどの文法記号も意味を持たない場合は$\$i$ は付かない。

このように EDSD で定義された言語に対しては手続きΔを次のように作成すればよい。

手続きΔの内部状態

point プログラムの現在の実行位置を示す変数。

mode プログラミング環境のどのような機能を使用しているかを示す変数で *nothing*、*trace*、*setBreakPoints* のいずれかの値を取る。

argumentList プログラムの実行途中での情報表示のために使用する変数。

手続きΔのアルゴリズム

実行・トレース・デバッグ対象にしている非終端記号の意味が $D_1 \rightarrow D_2 \rightarrow \cdots \rightarrow D_q$ の型を持つ意味関数で与えられるので、定数 $ARGNUM$ の値は $q-1$ にする。

```
procedeure Δ(M):
begin
  if Mが(λ{A'}x.L)K...の形をしている then
    begin
      A'の中の位置情報pをpointに入れる;
      if execute ∈ A' then
        begin
          for argumentCounter := 1 to ARGNUM do
            Mは(λ...)K...の形をしているはずなので、
            argumentListの最後にKを追加し、
            Mを最左戦略で1回簡約した結果を新しいMとする;
          if mode = trace then
            display(argumentList)
          else if mode = setBreakPoints then
            if pointがブレークポイントである then
              display(argumentList);
          argumentListを空にする;
          return M
        end
    end
  else
    begin
      Mを最左戦略で1回簡約した結果を新しいMとする;
      if M = DERROR then
        動的誤りが起こったことをユーザに報告し、
        誤りの診断をする
      else
        return M
    end
end
```

手続き *display* は *argumentList* の各要素で表示方法が定義されているものをウィンドウを使って表示する。もしユーザが実行の継続を指定すればΔに実行を戻して処理を続ける。

ユーザの入力したプログラムに字句・構文誤り、静的意味誤りがなければ実際にプログラムの実行・トレース・デバッグを行なうことができる。ユーザはプログラムの入力ファイルなどの初期状態をウィンドウを使って入力する。プログラムの動的表示的意味を初期状態に適用したラムダ式を生成し、それを簡約することがプログラムの実行に相当する。

実行

単にプログラムの実行結果だけが知りたい場合は、Δの *mode* 変数を *nothing* にセットしてから手続き ALTR を実行する。プログラムが終了する場合は正規形に簡約される。その正規形がプログラムの終了状態なので、それをウィンドウを使って表示してユーザにプログラムが終了したことを示す。

トレース

プログラムの実行トレースを行なう場合はΔの *mode* 変数を *trace* にセットしてから手続き ALTR を実行する。このとき、Δを

$$((\lambda\{execute, p\}\$d_1 \cdots \$d_{q-1}.M_d)N_1 \cdots N_{q-1})$$

の形をもつラムダ式を引数として実行した場合には以下の処理が行なわれる。まず *point* 変数に p が代入される。次に属性が *execute* を含むので *argumentList* に N_1 から N_{q-1} を順番に追加し、*argumentList* を手続き *display* で表示する。以上の動作は、プログラムテキストの p の位置にあるデバッグ対象のプログラムテキストに実行が到達したときのプログラムの状態や環境を表示したことに相当する。

5 おわりに

表示的意味記述からのプログラミング環境の生成方法について提案した。プログラミング言語の形式的定義をもとにプログラミング環境を構築しているので厳密で理論的なプログラムの作成・デバッグ支援をユーザに提供することができる。本システムは Smalltalk-80 で実現した。現在さまざまなプログラミングパラダイムをもつ言語への適応性を検証している。

参考文献

[1] J.E.Stoy. *Denotational Semantics — The Scott-Strachey Approach to Programming Language Theory*, The MIT Press, 1977.

[2] J.R.Hindley and J.P.Seldin. *Introduction to Combinators and λ-calculus*, Cambridge University Press, 1986.