5 E − 1

オブジェクトを用いた計算モデルとその2次元トレーサへの応用*

小西 善二郎†
早稲田大学大学院理工学研究科

二村 良彦‡
早稲田大学理工学部

1 始めに

トレーサはデバッギング・ツールの一つとして使われている。トレーサを一言で説明すれば、計算の途中の処理内容を出力するインタプリタと言える。ところが、出力される内容や形式、タイミングなどはトレーサの設計に依存し、必ずしもユーザにとって扱いやすいものではない。

これを解決する一つの方法は、インタプリタのソース・コードをユーザに与え、ユーザにそれをトレーサへ改造させることである。この場合、ユーザは与えられたインタプリタを解読する必要があり、たいていそれは困難な作業である。そこで著者は、新しく計算モデルを用意し、そのモデルに基づいてインタプリタを作成することを考えた。ユーザは、そのモデルを理解すれば簡単にインタプリタを解読でき、そしてそれをトレーサに改造できるのである。

本稿では、まず一般的に計算モデルを説明する。続いて具体例として、ある小さな Lisp [5] のインタプリタを CLOS [2, 3] で書く。最後に、トレーサへの改造の例として、2 次元トレーサ [4] を設計する。

2 計算モデル

インタプリタが出会う計算要素には、組み込みの／定義された関数名／手続き名や、変数名、定数、制御用キーワード、及び関数や変数の値などがある。これらは何らかの文脈のもとに出現する。本稿では、計算要素の出現とその文脈との関係を次のように考える。「計算要素は、ある計算要素の過去のある出現が予約したある文脈のもとに出現する。この出現はその直後、その計算要素の文脈を確定し、その出現自身が期待する、計算要素の未来の出現の文脈を予約する。」

例えば、Lisp のインタプリタが次のように動いたとする。

```
META-EVAL[1] (setq x '(a b))
(A B)
META-EVAL[2] (setq y '(c d))
(C D)
META-EVAL[3] (cons (car x) (cdr y))
(A D)
```

そして、インタプリタは 3 番目の式を計算していて、関数名 car に出会ったとする。この関数名は、関数名 cons の出現が予約した、引数の計算という文脈のもとに出現している。この car には、cons の出現における文脈の引数の関数名という文脈が与えられる。そして car の出現は、その引数に関する計算要素の出現とその関数全体の値の出現を期待し、その引数の計算、及びその値という文脈を予約するのである。

さて、上記の考え方で曖昧であった「文脈」というものを、予約の連なりと思うことにする。つまり、文脈を予約するのではなく、予約そのものが生じるのである。また、文脈は予約の末に確定もするので、予約の連なりの最後に確定を追加したものでもよいとする。予約と確定をシンボルで表せば、文脈はシンボルの列になる。

上記の例に戻り、インタプリタが 3 番目の式を計算し始めるところを考える。インタプリタが始めに出会うのは関数名 cons である。この出現は、第 1 引数の計算、第 2 引数の計算、及び全体の値を予約する。そこで、この cons に対する文脈をシンボル operator で表し、それらの予約をそれぞれシンボル arg1., arg2., value. で表す。続いてインタプリタは関数名 car に出会う。これは、arg1. という予約のもとに出現し、この予約の関数名という確定された文脈を持つ。そこで、この car の文脈を arg1.operator と表す。また、さらなる予約もあるので、それらを arg1.arg., arg1.value. と表す。これを続けていって、インタプリタが 3 番目

*A model of computation with objects and its application to the two-dimensional tracer

†Zenjiro Konishi. Graduate School of Science and Engineering, Waseda University. 4-1, Okubo 3 chome, Shinjuku-ku, Tokyo 169, Japan.

‡Yoshihiko Futamura. School of Science and Engineering, Waseda University. 4-1, Okubo 3 chome, Shinjuku-ku, Tokyo 169, Japan.

の式の計算を終えたなら、文脈と計算要素の対応は次のようになる。

- operator = cons
  - arg1.
    - operator = car
    - arg.
      - variable = x
      - value.return = (a b)
    - value.return = a
  - arg2.
    - operator = cdr
    - arg.
      - variable = y
      - value.return = (c d)
    - value.return = (d)
- value.return = (a d)

すべての計算要素には出現に応じて固有の文脈が与えられる。また、一度確定した文脈は二度と変わらない。

以上より、インタプリタを次のような規則の集まりと見なせる。「過去の計算要素とそれらの文脈をもとに、どのような計算要素を出現させ、どの予約に対して、その計算要素をどう確定させ、さらにどのような予約を発生させるか。」

この計算モデルに従ったインタプリタを実装するには、オブジェクト・システムを用いるのが適切である。つまり、出現をオブジェクトに、予約と確定をスロットに対応させ、規則をメソッドで書くのである。

3 Lisp インタプリタ

具体的に Lisp インタプリタを設計する。特別な処理が必要なのは、環境の取り扱いと、トップレベルの実現である。参照木を次のようにする。

- dummy = dummy
  - toplevel.
    - read = (setq...
    - eval.
      - operator = setq
      - arg1.···
      - ···
    - print.···
    - next.···
  - environment.
    - dummy = dummy
    - lexicalvars.···
    - specialvars.···
    - defuns.···

環境の部分にはフレーム [1, 5] が必要となるが、消滅を表すスロットと次の発生を表すスロットを持つクラスを用意すれば実現できる。

次に、インタプリタを改造して、2 次元トレーサを作る。出力される図式は、上記の参照木を横にして、value.return の内容を持ち上げる、といった単純な変換でできる。図式中の計算要素の配置は、その出現と同時に決まるので、計算と並行して図式が書ける。

```
TRACER[3] (cons (car x) (cdr y))

                          (a d)
---- --- ----- --- -----
cons       a          (d) |
---- --- ----- --- -----
     car (a b)|cdr (c d)|
---- --- ----- --- -----
           x  |       y  |
```

4 終わりに

与えられたインタプリタのソース・コードを解読する場合、問題なのは、ユーザに必要なデータがどの変数に格納されるかを探さなくてはいけないことである。たとえ見つかったとしても、データが格納され、また失われるタイミングを見定めなくてはいけない。

本稿の計算モデルにしたがったインタプリタなら、出現する計算要素がすべて一つの参照木にまとめられるので、容易に変数 (スロット) が見つけられる。また、一度参照木に割り当てられた計算要素は、失われたり他の要素に置き変わったりしないので、ユーザが改めて要素を保存したり、参照するタイミングを気にしたりする必要もない。

これらの特徴により、このように作られたインタプリタには、汎用性の高いデバッギング・ツールとしての利用が期待される。

参考文献

[1] Abelson, H., G. J. Sussman and J. Sussman: *Structure and Interpretation of Computer Programs* (2nd ed), MIT Press, 1996.

[2] Graham, P.: *ANSI Common Lisp*, Prentice-Hall, 1996.

[3] 井田昌之, 元吉文男, 大久保清貴 (編): Common Lisp オブジェクトシステム — CLOS とその周辺 —, bit 別冊, 共立出版, 1989.

[4] 小西善二郎: 関数型言語のための 2 次元トレーサ, 日本ソフトウェア科学会第 14 回大会論文集, 1997, pp. 21–24.

[5] Winston, P. H. and B. K. P. Horn: *Lisp* (2nd ed), Addison-Wesley, 1984.