5K-6

Object Oriented Language Fumi の作成とその評価について

吉田 洋一　　武市 正人
(電気通信大学計算機科学科)

0はじめに

われわれは、三菱電機の MELCOM COSMO 800III UTS/VS および MELCOM 70/250 VOS 上に、Object Oriented Language Fumi を開発した。これは、よく知られている Smalltalk-80 をもとに、文法を変更し、機能を拡張したものである。ここでは、Fumi の特徴と実現法について報告し、その評価を行う。

1 Fumi の特徴

1.1 文法の変更

Fumi の文法は、Smalltalk-80 のものに準拠している。しかし、いくつかの点で変更を加えた。

i) unary selector の変更

Smalltalk-80 では、unary selector は名前であったが、Fumi では、名前の前に '!' をつけることにした。これは、プログラムの読みやすさの観点から決定した。

ii) 代入文の変更

Smalltalk-80 では、

```
a<-b<-1
```

という形の代入文が許されているが、Fumi では、これを禁止した。これは、このような代入文が、Object Oriented Language 本来の考え方には合わないものであると考えたからである。上記の代入文は、Fumi では次のようにあらわされる。

```
a:=1
b:=1
```

iii) pool 変数の削除

大域変数は、method の中から頻繁に使用しない方が好ましい。しかし、クラスなどのように、どの method からも参照される大域変数も少なくない。そこで、pool 変数を用いることはやめて、クラスの中から参照される大域変数はクラスの中で宣言することにした。これによって、method の中で参照している変数は、すべて明確に宣言されることになる。

iv) block の変更

block に対する引数の受け渡しをなくした。このように機能を縮小した理由は、block の実行に関する実現上の問題点と、意味上の問題点が多いためである。

1.2 機能の拡張

i) 型の検査

代入と引数の受け渡しにおける型の検査を選択的に行えるようにした。ただし、大域変数については、クラスの保護のため、常に型の検査をする。型の検査は、コンパイル時の method の探査のためではなく、実行時のデバッグのために使うことを目的としたものである。また、1つの変数に対して可能な複数の型の宣言を許すことによって、Fumi の柔軟性をそこなわないようにした。型の階層も考慮して、検査を行うようにした。すなわち、

```
instance var x is integer + float
```

は、

```
instance var x is number
```

と書くこともできる。

ii) 環境の取り込みと保存

Fumi は、対話的に使用されるので、実行時に作られた環境を保存しておく必要がある。環境の保存は、ゴミ集めで詰め合わせをし、そのときの記憶領域にある環境をファイルに出すようにした。このときの形式は、ふたたび記憶領域に取り込むときに、どの位置にも置くことができるように、再配置可能のものにしている。

2 Fumi の実現法

Fumi は BCPL とアセンブラで書かれておりどちらも約 4000 行である。また、構文解析は、Yacc を用いた(規則数は59)。

以下では、実現上で、特徴的なことがらについて述べる。

2.1 静的な環境

Fumi の静的な環境は、すべて大域辞書から参照できる。Fumi の中に存在する object の中で、大域辞書を通して参照される変数(クラス、大域変数、クラス変数)は、すべて dictionary item という変数名とその実体を属性としてもつ object を通して参照される。こうして、辞書を用いて、1つの変数に対して、その実体を一意に決める方法を実現している。これにより Fumi では、変数名とその実体との結合が、クラスのコンパイル時に可能となり、環境の時間的経緯に左右されることはない。ただし、クラス変数の場合には例外

的なことも起りうる。

大域辞書とmethod辞書には、ハッシュ法を用いた。ハッシュ法の効率向上のため、表の大きさは、登録されている項目数の2倍以上になるようにしている。大域辞書の場合、登録される変数の数が増加するが、表の半数を占めると、ハッシュのやり直しをする。

2.2 Method探査

method patternは、ハッシュ法を用いて、1つの表で一括して取り扱う。コンパイル時にmethod patternを表に登録し、環境には、その表の指標を登録しておく。実行時には、この指標を用いてmethod探査を行う。また、objectはそれが属しているクラスのdictionary itemを持っているので、mehtod探査時には大域辞書を参照することはない。

2.3 継承

クラスの定義においてsuper classが宣言されたとき、次の規則にしたがってクラスの変数が継承される。

i) super classのクラス変数は、無条件で継承される。

ii) instance variableとindexed variableのどちらの宣言もない場合は、super classのものが継承される。

iii) instance variableの宣言がある場合には、名前の宣言されている順によって変数の継承が行われる。

method探査における継承の効率向上のために、クラスのobjectの中にsuper classのdictionary itemをもつことにした。これにより、実行時に大域辞書を参照する必要はなくなる。

2.4 記憶管理

Fumiで用いられているゴミ集めの方法は、記憶領域を二等分して、片方が一杯になれば、もう一方に詰め直おすという方法を用いた。これは、共有される可変長のセルが数多く存在し、また記憶領域にもゆとりがあることを考慮して決定した。

3 評価

Fumiが完成してから、画面エディタなどの応用プログラムを作成した経験から得られたいくつかの知見を述べることとする。

Object Oriented Languageであるシステムを作成する際には、最初にそのシステムの概念図を書くことに始まり、その概念図に忠実な形でプログラムを書き下すことができるという、ほかの言語にはない特徴を持っていると考えられる。この手法を用いて、非常に短期間で画面エディタを作成することができた。また、対話的に作業ができ、システムに対するコマンドもFumiの式であるということも大きな要因であると考えられる。しかし現在のFumiでは、中間コードを逐次解釈しているため、速度の面では問題が残っている。機械語を生成して速度の向上を計ることが今後の課題と考えている。

Reference

i) Goldberg and Robson, SMALLTALK-80 THE LANGUAGE AND DESCRIPTION, ADDISON WESLEY

ii) Goldberg and Robson, Special Issue on Smalltalk, In BYTE Magazine, , August 1981