

関数的プログラム作成支援システムのための構造エディタ

永田 守男 折田 圭子*
(慶應義塾大学工学部)

1. はじめに

マイクロコンピュータから超大型機に至るまで、使う計算機の機種やその使用目的は様々であるが、オンラインの端末の前に座って仕事をするという利用形態が多くなってきた。その結果、計算機を使う毎にエディタの世話になることになり、エディタの使い心地の如何がプログラムに与える心理的な影響は甚大で、計算機を直接的に利用している人のほとんどがエディタについての一家言を持っており、いつても言い過ぎでない程である。比較的古くからある普通の形の

・テキストエディタ

をはじめとして、

・構造エディタ

・スクリーンエディタ

などと呼ばれる種類のものも増え、それぞれに「使い易さ」を競いあっている。

こうした状況にあって、筆者たちの開発してきたKSRシステム(Keio Support system for functional Recursive programming) [4] が「会話型」を標榜しているにもかかわらず簡単な「行エディタ」しか準備していないのは不十分であったので、今回「構造エディタ」を新たに作成した。

この新しいエディタ(KSRエディタと呼ぶ)は、有名なINTERLISPエディタ [6] の考え方に基づいて設計・製作したが、次のような特徴を持つ。

(1) KSRシステムの扱うM表現に近い形のものを編集の対象にしている。

(2) 簡単で統一納な命令体系になっている。

(3) ミニコンのLISP上で実現できる程度の大きさである。

さらに、構造エディタであるから、このKSRエディタを使えば、利用者の入力や修正によって一定の構文を崩さないことは勿論である。

ここでは、こうした特徴を中心にして、KSRエディタを例を使って説明する。KSRシステムについては、この論文ではその概要を必要最小限示すだけにとどめるので、詳細は参考文献 [3, 4] を参照していただきたい。

2. KSRシステムの概要

KSRシステムの目的、役割および利用者の入出力の形を例をあげて述べ、KSRエディタが編集の対象とするものを具体的に説明する。Lisp言語のM表現に近い形のものを想定した記法を考えているが、その他の関数的なプログラミング言語 [1, 2 など] も念頭に置いている。すなわち、こうした言語の形式的な特徴として、

*) 現在、日本電気(株)

(Ⅰ)関数の合成

(Ⅱ)マッカーシーの条件式

(Ⅲ)再帰呼び出し

の三つを規定する。言語をLispに限定しないで、こうした種類の言語でプログラムを書くときに人間の見落としやすい論理的なミスなどを機械の方で検出し、会話的な形でプログラム開発を容易にするのがKSRシステムの目的である。プログラムが完成してから虫を取るのではなく、'プログラム開発時に誤りの少ないものを作る'ことを助けるのが本システムであって、新しく関数的言語を提案するというようなものではない。

KSRシステムは、大きく分けて二つの部分からできていて、それぞれKSR1とKSR2と呼んでいる。KSR1は、開発中のプログラムの論理的な整合性を調べ、KSR2がこのプログラムの停止性を検査する。例を使ってこれらを説明する。

いま、二つのアトムどうしの同等性を調べるEQという関数があるときに、一般的な2進リスト(BLISTと呼ぶ)の同等性を調べるEQUALを定義することを考える。左端に'/'のある行が利用者からの入力で、その他の行はシステムからの出力である。まず、KSR1システムを動かして、EQUALがBLISTという型の2引数を持ち、値がBOOLEAN型の関数であることを示す会話例は次のようになる。ここでHDとTLは、2進リストのそれぞれheadの部分とtailの部分をとる関数である。

```
/ KSR1()
```

```
.... KSR1 ....
```

```
COMMAND MODE
```

```
/ DEF
```

```
WHAT IS FUNC NAME?
```

```
/ EQUAL
```

```
INPUT FUNC SPEC
```

```
/ EQUAL[X.BLIST:Y.BLIST],BOOLEAN
```

にだし、最終の行にある関数の型のあとに、保護命題(guard proposition)と呼ぶものが書ける。これは、引数に対して更に条件を付けるときにだけ使う。こままでを仕様部(specification part)と呼ぶ。

このあとEQUALの定義本体を入力する。例を示すと、

```
INPUT FRAGMENT
```

```
/ < ATOM[X]&ATOM[Y] -> EQ[X;Y] >
```

```
/ < \ATOM[X]&ATOM[Y] -> F >
```

```
/ < \ATOM[X]&\ATOM[Y] -> EQUAL[HD[X];HD[Y]]&EQUAL[TL[X];TL[Y]] >
```

```
/ END
```

のようになるが、このときはシステムから、

MISSING CONDITION
ATOM[X] & \ATOM[Y]

というメッセージが出てくる。この条件の場合を考え落としているというのである。ここで ' $<$ ' と ' $>$ ' でくくられたものを フラグメント (fragment) と呼び、この中の ' $\rightarrow$ ' の左側を条件部 (condition part)、右側を式部 (expression part) と呼ぶ。仕様部といくつかのフラグメントを合わせて一つの関数を定義するものを F-プログラム と名づける。また、EQ という関数の二つの引数が共にアトム有的时候のみ EQ は有効であるという情報が入力してあると、もしも第1番目のフラグメントの条件部を

ATOM[X] & \ATOM[Y]

のように間違えて入力すれば、一番目のフラグメントの入力時点でこの誤りは機械的に検出される。

KSR1 には簡単な定理証明プログラムがあって、こうした論理的な整合性を調べることができる。フラグメントのネスト構造が複雑になったときなどには、このような機能があると便利である。

KSR2 に対しては、利用者が 底命題 (bottom predicate) というものを入力しておく。たとえば、BLIST に HD または TL を繰り返し適用すれば、ATOM という述語が必ず真になるというようなものである。この命題が入力してあり、前述の EQUAL の定義に

$\langle \text{ATOM}[X] \& \backslash \text{ATOM}[Y] \rightarrow F \rangle$

が追加されていれば、KSR2 は EQUAL が停止することを保証する。停止性を保証できないときには、その旨を利用者に知らせようになっている。

以上に示したように、本システムでは、会話的な利用の特性を生かして、フラグメント単位または関数単位のきめ細かなプログラミング補助を行なっている。なお、本システムで使っているアルゴリズムならばその正当性については [3] を参照されたい。

3. KSRエディタの設計

ここまで述べてきたようなものを編集するための構造エディタを作るわけであるが、編集の対象となるものを大別すると、F-プログラムおよび補助的な情報の二つである。補助的な情報とは、底命題や論理的な諸性質 (たとえば x が整数のときに ' $\neg(x < 0)$ ' でかつ ' $\neg(x = 0)$ ' ならば ' $(x > 0)$ ') のようなものを指すが、ここでは主に F-プログラムを編集することが多いので、本論文の以降では F-プログラムについてのみ考えることにする。

3. 1 F-プログラムの構造とポインタ

F-プログラムは、ひとつの仕様部といくつかのフラグメントからできている。

仕様部は一定の規則に従った構造になっている、フラグメントも条件部と式部という構造になっている。ただし、フラグメントの方は、ネストになることを許している。

F-プログラムがこのような構造に従っているうえに関数的な形なので、関数名と引数のペアが基本的な形である。また、引数にこうしたペアがくるようなネスト構造になったものもある。

したがって、Interlispエディタに則ってポインタと現在式(current expression)の概念を取り入れるためには、上述の構造に適った命令体系を考えればよい。

3. 1. 1 挿入・削除・置き換えおよび印刷

KSRエディタは構造エディタであるから、現在式を対象にしてその編集をおこなうが、ポインタの移動などの説明の前に、あとの例題で使う程度の挿入、削除、置き換えおよび印刷に関する命令をまとめておく。

(1) 挿入 I ± n [< exp >]

現在式の n 番目のものの後(+)のとき)または前(-のとき)に < exp > を挿入する。± と n が省略されていると、それぞれ +, 0 とみなす。

(2) 削除 D

現在式を削除する。

(3) 置き換え C [< exp >]

現在式を < exp > で置き換える。

(4) 印刷 P

現在式を印刷する。

これらの他に、ポインタを1レベル上に戻す 'U'、また、トップレベルに戻す '↑' などがあるが、「レベル」などについてはこの後に説明する。印刷のときにトップレベルにする '!' もある。'UNDO' (ひとつ前の命令を取り消す) も全体にわたって使える。なお、数字 (n) を書くと、 n 番目の式にポインタが移る。サーチ命令として 'S [< exp >]' があって、< exp > にポインタを移動できる。

3. 1. 2 仕様部の修正

'S' という命令で仕様部を現在式としたとき、これを修正する命令について、例をあげて説明する。

前に示したように、仕様部は、関数名と引数のペア、この関数の値の型、保護命題の三つの部分からできている。そこで、次のようなポインタ移動の命令がある。

F ... 関数名へポインタを移す。

- A ... 引数のところへポインタを移す。
- FT ... 関数の値の型へポインタを移す。
- G ... 保護命題へポインタを移す。

これらを使った修正例を次に示す。

```
A[X.IMT;Z.INT],INT,\MINUSP[X]
```

を次のように変更するための会話例である。

```
ACK[X.INT;Y.INT],INT,\MINUSP[X]&\MINUSP[Y]
```

ここで‘ED /’に続くのが利用者の入力部分である。

```
ED/
A[X.IMT;Z.INT],INT

ED/ F C[ACK] U
ACK[X.IMT;Z.INT],INT

ED/ A 1 T C[INT] U U
X.INT;Z.INT

ED/ 2 N C[Y] ↑
ACK[X.INT;Y.INT],INT

ED/ B
EXIT SPMN

ED/ G
\MINUSP[X]

ED/ I+1[\MINUSP[Y]]
\MINUSP[X]&\MINUSP[Y]

ED/ B
EXIT GUARD

ED/ B
EXIT SPEC
```

3. 1. 3 フラグメントの修正

‘P’ という命令でフラグメントの集合へポインタを移す。各フラグメントは必ず

< 左辺 → 右辺 >

の形をしている。ただし、左辺と右辺のところにはフラグメントがくることがある。したがって、ポインタの移動としては次の二つを用意すれば十分である。

- L ... 現在式が<左辺→右辺>の形のとき左辺にポインタを移す。
- R ... Lと同様で右辺にポインタを移す。

次のようなフラグメントの集合

```

< ZEROP[X]
-> Y >
< \ZEROP[X]&ZEROP[Y]
-> ACK[SUB1[X];1] >
< \ZEROP[X]&\ZEROP[Y]
-> ACK[ACK[SUB1[X];Y];SUB1[Y]] >

```

を修正して

```

< ZEROP[X]
-> ADD1[Y] >
< \ZEROP[X]&ZEROP[Y]
-> ACK[SUB1[X];1] >
< \ZEROP[X]&\ZEROP[Y]
-> ACK[SUB1[X];ACK[X;SUB1[Y]] >

```

のように変更する会話例は次のようになる。

```

/ P
< ZEROP[X]
-> Y >
< \ZEROP[X]&ZEROP[Y]
-> ACK[SUB1[X];1] >
< \ZEROP[X]&\ZEROP[Y]
-> ACK[ACK[0;Y];SUB1[Y]] >

```

```

ED/ 1
< ZEROP[X]
-> Y >

```

```

ED/ R
Y

```

```

ED/ C[ADD1[*]]
ADD1[Y]

```

```

ED/ U
< ZEROP[X]
-> ADD1[Y] >
< \ZEROP[X]&ZEROP[Y]
-> ACK[SUB1[X];1] >
< \ZEROP[X]&\ZEROP[Y]
-> ACK[ACK[0;Y];SUB1[Y]] >

```

```

ED/ 3
< \ZEROP[X]&\ZEROP[Y]
-> ACK[ACK[0;Y];SUB1[Y]] >

```

```

ED/ S[ACK]
ACK

```

```

ED/ U
ACK[ACK[0;Y];SUB1[Y]]

```

```

ED/ A
ACK[SUB1[X];Y];SUB1[Y]

```

```

ED/ 1
ACK[SUB1[X];Y]

```

```

ED/ C[SUB1[X]]
SUB1[X]

```

```

ED/ U
SUB1[X];SUB1[Y]

```

```

ED/ 2
SUB1[Y]

ED/ C[ACK[X;*]]
ACK[X;SUB1[Y]]

ED/ U
SUB1[X];ACK[X;SUB1[Y]]

ED/ +
< ZEROP[X]
    -> ADD1[Y] >
< \ZEROP[X]&ZEROP[Y]
    -> ACK[SUB1[X];1] >
< \ZEROP[X]&\ZEROP[Y]
    -> ACK[SUB1[X];ACK[X;@]] >

ED/ B
EXIT PRG

```

4. KSRエディタの製作と評価

以上のようなエディタを、PDP 11のKLISP（自由セルが約10K）上に試作し、使用してみた。前節で示した例題と同じことを実行した会話例を付録につけておくので、全体の感じについては付録の方を参照していただきたい。KSRエディタの実際の製作にあたっては、ミニコンのLispを使うこと、KSR1とKSR2との関係をつけることなどのために工夫した箇所がいくつかある。こうした点と、試用を通してのKSRエディタの評価などについてまとめておく。

4. 1 製作

KSR1とKSR2が既にKLISP上で動いているので、このKSRエディタもKLISPを使って製作した。したがって、内部形としてこれらが対象とするものはLISPのS表現になっている。一方、KSRシステムはその他の関数的な言語のことを考慮し、これまでに示してきたような形での入出力を通してシステムと人間とが会話するように考えてある。利用者の入力からKSRエディタが何らかの処理をして結果を表示する間の関係を図に示すと次のようになる。

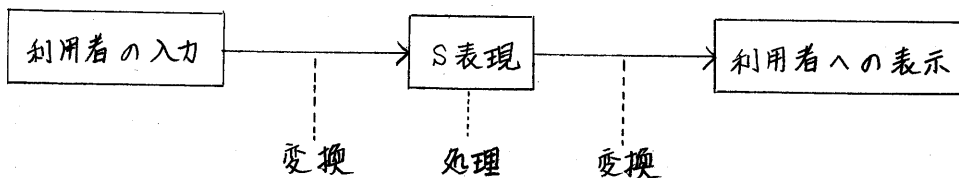


図1 利用者との入出力と内部形

これらの変換を行なう部分以外は、普通のLispエディタと同じようにして作ればKSRエディタができ上がる。PDP 11/21上で作ったKSRエディタは、KLISPのソースリストとして約1000行のシステムになったが、このうち多く

の部分全体が全体の制御とこうした変換のためのもので、処理そのものは約3割強の部分しか占めていない。

4.2 評価

こうしたシステムの評価というのは難しいが、「使いやすさ」について反省を加えてみる。使うときに気になる要因としては「命令体系の良さ」と「応答時間」があるので、この二つの側面から考える。

まず応答時間について。これはガーベッジコレクションのこともあるので一般的なことは言えないが、10~20行程度の関数を同時にいくつか入れておく程度のことならば、1~2秒で応答が返ってくる。ただし、サーチ命令を実行すると5秒ぐらい待つこともある。いずれにせよ、イライラするほどの時間ではない。

次に命令体系について。これも多勢の人に使うてもらうことをしていないので十分な評価はできないが、当初懸念していたよりも使い良いものが作れたようである。これは、編集の対象となるF-プログラムの構造が簡潔なために、命令体系がすっきりした形になったからである。最近いろいろと研究されている関数的または述語的プログラミング言語のようなものであれば、ここに示したような構造エディタは十分に役立つものと思われる。また、スクリーンエディタなどの長所も取り入れたりすることも考えてよい。

KSR1, KSR2, KSRエディタを作成してみても、こうしたシステムと言語処理系がひとつになって図2のようなものができると、利用者にとっては便利になると思われる。

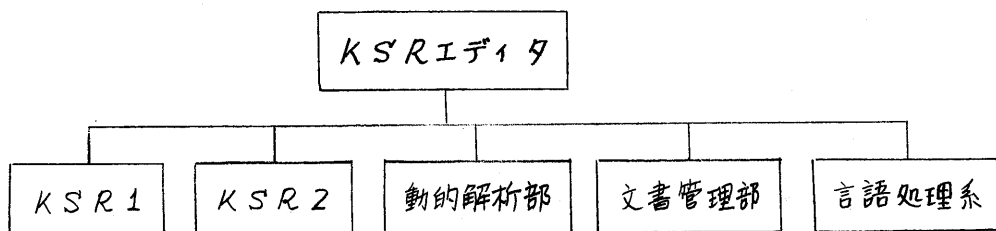


図2 統合的なプログラミングシステム

5. おわりに

いろいろと構造エディタが作られている[6, 7など]が、関数的な言語でM表現に近い形のものについては簡潔で役に立つものができると分かった。これからのソフトウェアにはKSRシステムのような言語と人間のインタフェースになるものが不可欠であろう。

参考文献

- [1] Backus, J.: Can programming be liberated from the von Neumann style? A functional style and its algebra of programs, CACM, Vol. 21, No. 8, 1978, pp. 613 - 641
- [2] Landin, P.J.: The next 700 programming languages, CACM, Vol. 9, No. 3, 1966, pp. 157 - 164
- [3] Nagata, M.: Interactive debugging for functional recursive programming, 京都大学数理解析研究所研究報告 396, 1980, pp. 131 - 169
- [4] Nagata, M., Akiyama, T. and Fujikake, Y.: An interactive supporting system for functional recursive programming, Proc. IFIP 80, 1980, pp. 263 - 268
- [5] Teitelman, W.: Interlisp Reference Manual, Xerox PARC, 1974
- [6] Wilander, J.: An interactive programming system for Pascal, BIT, Vol. 20, 1980, pp. 163 - 174
- [7] 車谷 ほか: プログラム作成支援システム KSP の構造と設計, ソフトウェア工学研究資料, 1981年2号

付録

第3節で説明のために断片的に示した会話例のほとんど同じものの全体である。アッカーマン関数の仕様部とフラグメントの一部が誤って入力されているときの修正例である。KLISP上でKSRシステムをロードしてるところから示す。

```

ARGS ARE,
/ EDIT2()

F(PROG,P(RED,B(YE?
/ F

WHAT IS FUNC NAME?
/ A

S(PEC,P(ROG,B(YE?
/ S

A(LL,M(SPC,G(UARD,B(YE?
/ N
.... ILLEGAL COMMAND

A(LL,M(SPC,G(UARD,B(YE?
/ M
~,U,F,A,FT,N,T,number
I,D,C,P,!,*,B
ACX.INT;Y.INT],INT

ED/ FCCACK]
ACK

ED/ U
ACKEX.INT;Y.INT],INT

ED/ FT
INT

```

```

ED/ [CLIST]
LIST!INT

ED/ UNDO
INT

ED/ U
ACKEX.INT;Y.INT],INT

ED/ A
X.INT;Y.INT

ED/ 2
Y.INT

ED/ 1
INT

ED/ 11
Y.INT

ED/ 7
ACKEX.INT;Y.INT],INT

ED/ 8
EXIT SPMN

A(LL,M(SPC,G(UARD,B(YE?
/ G

```

(次ページ左へ)

```
number,F,A,U,^
I,D,C,P,!,*,B
\MINUSPCX] & \MINUSPCY]
```

```
ED/ 2
\MINUSPCY]
```

```
ED/ A
MINUSPCY]
```

```
ED/ A
Y
```

```
ED/ C[ZEROPC*]]
ZEROPCY]
```

```
ED/ UU
\MINUSPCZEROPCY]]
```

```
ED/ UNDO
Y
```

```
ED/ ^
\MINUSPCX] & \MINUSPCY]
```

```
ED/ 8
EXIT GARD
```

```
A(LL,M(SPC,G(UARD,B(YE?
/ B
EXIT SPEC
```

```
S(PEC,P(ROG,B(YE?
/ P
NUMBER,L,R,F,A,U,^
I,D,C,P,!,*,B
< ZEROPCX]
->Y >
< \ZEROPCX] & ZEROPCY]
->ACK(SUB1CX]#1] >
< \ZEROPCX] & \ZEROPCY]
->ACK(ACKC@;Y]#SUB1CY]] >
```

```
ED/ 1
< ZEROPCX]
->Y >
```

```
ED/ R
Y
```

```
ED/ C[ADD1C*]]
ADD1CY]
```

```
ED/ U
< ZEROPCX]
->ADD1CY] >
```

```
ED/ U
< ZEROPCX]
->ADD1CY] >
< \ZEROPCX] & ZEROPCY]
->ACK(SUB1CX]#1] >
< \ZEROPCX] & \ZEROPCY]
->ACK(ACKC@;Y]#SUB1CY]] >
```

```
ED/ 3
< \ZEROPCX] & \ZEROPCY]
->ACK(ACKC@;Y]#SUB1CY]] >
```

```
ED/ S[ACK]
ACK
```

```
ED/ U
ACK(ACKC@;Y]#SUB1CY]]
```

```
ED/ A
ACK(SUB1CX]#Y]#SUB1CY]]
```

```
ED/ 1
ACK(SUB1CX]#Y]
```

```
ED/ C[SUB1CX]]
SUB1CX]
```

```
ED/ U
SUB1CX]#SUB1CY]]
```

```
ED/ 2
SUB1CY]
```

```
ED/ C[ACKCX;*]]
ACKCX#SUB1CY]]
```

```
ED/ U
SUB1CX]#ACKCX#SUB1CY]]
```

```
ED/ ^
< ZEROPCX]
->ADD1CY] >
< \ZEROPCX] & ZEROPCY]
->ACK(SUB1CX]#1] >
< \ZEROPCX] & \ZEROPCY]
->ACK(SUB1CX]#ACKCX#0]] >
```

```
ED/ 8
EXIT PRG
```

```
S(PEC,P(ROG,B(YE?
/ B
```

```
WHAT IS FUNC NAME?
/ END
EXIT FPROG
```

```
F(PROG,P(RED,B(YE?
/ B
EXIT EDIT
```

```
VALUE IS,
NIL
```

```
ARGS ARE,
/
```

(おわり)