2F-4 ライブラリモジュールを用いたプログラム生成における知識の利用について

藤田 米春　高松 忍　西田 富士夫
大阪府立大学

1．まえがき　階層的に定義されたライブラリモジュールを用いてプログラムを生成するシステムについて文献(1)において報告したが、この様なシステムにおいては次の点が問題になる。

a)モジュールは汎用的でなければならないのでデータの特殊性などを用いた最適なアルゴリズムを用いることができない。

b)モジュールは一種のプログラムであり、アルゴリズムの形に解かれた形の知識と考えられる。従って、アルゴリズムの形でない知識を記述することができない。

問題点a)から、モジュールの使用時にデータの特殊性などの知識を用いてモジュールを最適化することが必要であり、問題点b)から、モジュールの形以外の知識の利用法が必要となる。本報告ではこの様な知識をモジュールによるプログラム生成に利用する方法について述べる。

2．モジュール記述　モジュールの記述は見出し部と実現部とから成り、見出し部はPROC部、IN-OUT部、及びTYPE部から成る。PROC部は各引数に格名が付く他は、いわゆる手続きの頭部と同様の形式であり、IN-OUT部は入出力関係を格名付きの述語により記述したものである。TYPE部は変数の型の記述である。実現部(OP部)は、PROC部と同様の表現を制御構造を用いて組み合わせたものであり、いわゆる手続きの本体にあたる。表1にモジュールの一例を示す。モジュールのOP部には、?SELECT(q,q1:s1,...,qn,sn)の形の部分があり、システムの問い合わせにより、qの値をqiと指定するとsiによりこの部分が置き換えられる。これを用いてある程度の柔軟性をモジュールに持たせ得るが、より一般的な変更や最適化等を行なうことも必要になる。

3．仕様と詳細化　プログラムの仕様はモジュールのOP部と同じ形をしており、一種のプログラムである。詳細化は、仕様に現われるPROC表現とモジュールのPROC表現との単一化を試み、成功すれば単一化代入をモジュールのOP部に施して結果により仕様のPROC表現を置き換えるかPROC表現を手続き呼び出しの形に変更し、モジュールをPROCEDUREの形に変換する。これを繰り返して、詳細化が不可能になったらユーザが指定する目的言語に変換してプログラムを得る。

表1　表処理モジュールの例

```
 1 proc:#-retrieve(so:*x(*n1 .. *n2,*m1 .. *m2),
 2                 obj:*x(*i,*m1 .. *m2)
 3                  (cond:*p(*x(*i,*j),*r(*j))),
 4                 goal:*z(*n1 .. *n2,*m1 .. *m2)),
 5 entitytype:
 6  *x(reg:real,struct:array(*n1 .. *n2,*m1 .. *m2),
 7     vrole:input);
 8  *z(reg:real,struct:array(*n1 .. *n2,*m1 .. *m2),
 9     vrole:output);
10  *r(reg:real,struct:array(*m1 .. *m2),
11     vrole:input);
12  *p(reg:boolean,struct:pred,vrole:input);
13  '*n1,*n2,*m1,*m2'(reg:integer,struct:scalar,
14                    vrole:input);
15  '*i,*j'(reg:integer,struct:scalar,vrole:local),
16 in:given(obj:*x(*n1 .. *n2,*m1 .. *m2);*r(*m1 .. *m2)),
17 out:given(obj:#-retrieve(so:*x(*n1 .. *n2,*m1 .. *m2),
18                          obj:*x(*i,*m1 .. *m2)
19                           (cond:*p(*x(*i,*j),*r(*j))))),
20 op:?select(q1,
21     key-retrieve:
22      ?select(q2,
23       binery-retrieve:
24        #-if_then(cond:#-not(sorted(obj:*x(*n1 .. *n2,
25                   *m1 .. *m2),key:*x(*i,1) )),
26                  op:#-sort(obj:*x(*n1 .. *n2,*m1 .. *m2),
27                            key:*x(*i,1),
28                            goal:*x(*n1 .. *n2,*m1 .. *m2) ));
29        #-binary-retrieve(
30         so:*x(*n1 .. *n2,*m1 .. *m2),
31         obj:*x(*i,*m1 .. *m2)(cond: #-equal(*x(*i,1),*r(1))),
32         key:*x(*i,1),goal:*z(1,*m1 .. *m2) ),
33       sequential-retrieve: =(q1,general-retrieve)),
34     general-retrieve:
35      #-sequential-retrieve(
36       so:*x(*n1 .. *n2,*m1 .. *m2),
37       obj:*x(*i,*m1 .. *m2)
38        (cond:*p(*x(*i,*m1 .. *m2),*r(*m1 .. *m2))),
39       goal:*z(*n1 .. *n2,*m1 .. *m2))),
40  $
```

Application of knowledge to program generation by modules.

Yonebaru FUJITA, Shinobu TAKAMATSU and Fujio NISHIDA
UNIVERSITY OF OSAKA PREFECTURE

4．詳細化時における知識の利用　上記の様な詳細化において、最適化およびモジュールのOP部の変形が必要であるが、詳細化時に行なわれる単一化代入や仕様の入力条件によって定数や制限された値を引数とする関数や手続きには以下に述べるように、モジュールや一般的な知識を用いて最適化などを行なえる。

(a) 引数の一部に定数が与えられたモジュールに対する特殊化モジュールが存在する場合モジュールのOP部にCONDITIONAL(COND1:P1(x),OP1:S1,COND2:P2(x),OP2:S2,...,CONDn:Pn(x),OPn:Sn) の形の条件詳細化部を設け、引数xの定数化がPiを満たせば、モジュールSiによってこれを置き換える。

(b) 関数の引数全部が定数化され詳細化システムによって直ちに計算可能な場合には、システムがこの計算を行ない、結果によってこの関数を置き換える。

(c) 関数やモジュールについて、適当な定理等を用いてアルゴリズムを簡単化したり変形したりできる場合には、モジュールや関数の名前とこれに関係のある定理や公式等の対を表にしておき、詳細化時に用いて変形する。

[例1] 仕様が

```
FIND_STRING(SO:X(1..N),OBJ:I,COND:X(I,I+1-1)=S(1,1),GOAL:Z);
```

であり、FIND_STRING(SO:x(1..n),OBJ:i,COND:x(i,i+l-1)=s(1,l),GOAL:z) モジュールのOP部が、

```
OP:CONDITIONAL(
    COND1:l=1,OP1:FIND_CHAR(SO:x(1..n),OBJ:i,COND:x(i)=s,GOAL:z),
    COND2:l>1,OP2:FOR(COUNT:i,FROM:1,TO:n,
                      OP:IF_THEN(
                          COND:EQUAL_STRING(x(i..i+l-1),s(1..l)),
                          OP:z:=i;EXIT_LOOP(0) ) ) )
```

と成っている場合、仕様とモジュールのPROC部の単一化により l←1なる代入がされ、これにより CONDITIONAL文の第1条件が成り立ち、このOP部は FIND_CHARモジュールのPROC部により置き換えられる。

[例2] 関数PRIME(OBJ:x)のOP部において

```
y:=T;
FOR(COUNT:i,FROM:2,TO:x-1,OP:y:=y&mod(x,i)≠0);
```

となっており、モジュールの OUT部で

$$y \iff \mathrm{Any}(i:i\in(2..x-1))\,\mathrm{mod}(x,i)\neq 0$$

となっているとき、PRIME に関係する定理

$$\mathrm{Any}(i:i\in(2..x-1))\,\mathrm{mod}(x,i)\neq 0 \iff \mathrm{Any}(i:i\in(2..(\sqrt{x})))\,\mathrm{mod}(x,i)\neq 0$$

により x-1の代わりに(√x)を用いて

```
y:=T;
FOR(COUNT:i,FROM:2,TO:(√x),OP:y:=y&mod(x,i)≠0);
```

とする。

5．むすび　モジュールを用いてプログラムを生成する場合、上記の他に単位間の換算計数の表の様な知識も必要である。この様な表型の知識の取り扱いについても現在検討中である。

文献　(1) 西田、藤田：ライブラリモジュールを用いたプログラムの半自動的詳細化、情報処理学会論文誌Vol.25,No.5.pp.785-793(1984).