プロログによるモジュールの検索とプログラムの合成

恐神正博*　西田富士夫**

Prolog_based modeule retrieval and program_generation

Masahiro Osogami and Fujio Nishida

A new method of Prolog_based module retrieval and automatic program generation is proposed. A header table of modules for on_line retrieval consists of header name, comments that explain functions of the module in Japanese, argument type, function type, applied input condition, output condition, file_name and other attribute items of each library module.

The table is used for making machine readable refined specifications from manually written specifications. The program generating system MAPP produces C programs from the machine readable specification. MAPP can also make a C program from the input_output conditions of specifications by linking several modules.

1．まえがき

近年プログラムの再利用の研究が内外で盛んに行われている。これは過去に作成したプログラムやプログラムモジュールの中、現在与えられた仕様に適用できるものを再利用することにより、誤りの少ない、プログラムを短時間に作成しようとするものである。筆者らも基本的なプログラムモジュールを知識ベースとしてライブラリに収納しておき、これを利用してプログラムを自動合成する研究を行っており、実験システムMAPP(Module Aided Programming system by Prolog)の構築改良と拡張を続けている。この論文はモジュールの機能や処理対象のデータ構造や型などに関する属性項目を記述したモジュールの見出し表を作成し、仕様に対して適用可能なモジュールを検索しＣ言語のプログラムを合成する一つの方法と実験システムについて主として報告する。実験システムの処理はプロログで行っているため、記号列の変換やモジュールの検索やリンクなど、複雑な処理を比較的簡単に行うことができる。この手法は、ＣＯＢＯＬなど他のプログラム言語への展開などにも有用である。

2．モジュール見出しの属性項目

図１はモジュール見出し表における主要なモジュール属性項目記述の例である。各見出しはモジュール毎にmodule(H,T).の形式をとり、HやTの引き数部はリスト形式をとる。Hはモジュールの機能の種別を表す大見出しや、モジュール番号などからなる。Tは以下のようないくつかの関数形の項目からなる。これらは、モジュールの関数の型と機能を日本語文で説明するコメント項目、この関数をＣ言語プログラムでコールするときの関数表現の項目、引き数部の処理対象の変数の型や関数の戻り値の型を記述するタイプ項目、このモジュールを適用するのに必要な入力データやその性質などの条件を記述する入力項目、適用後、出力するデータの名前や性質を記述する出力項目、このモジュールを収納するファイル名項目などである。これらの属性項目の記述はリストを用いているため不定長でよく、また属性項目の種類はモジュールを通じて同じである必要はない。

モジュールの見出し項目にはこの他、３節でふれるように、(1)関数値のint,charなど関数の型名を記録する関数値タイプ項目、(2)乱数発生モジュールのように、発生乱数の下限、上限、個数のよ

＊事務局　＊＊経営工学科

うなパラメータ的変数の名前と型を記録するパラメータ変数項目、(3)配列要素の和や平均を求めるモジュールのように、求めた関数値を置数し、予め定めたルールにより命名した名前をもつ一時置数変数の名前と型を、モジュールの自動リンクのために、記録する一時置数変数タイプ項目などがある。

```
module([読み込む,1],
[com(大きさNの浮動小数点型１次元配列Aにキーボードからデータを読み込む),
funct(read_lar_df(A,N)),
type([float([A,200]),int(N)]),
in(read(N)),
out(read[[A,N]]),
file_name(readladf)]).
```

＜図１＞

目的プログラム言語の命令やマクロ、たとえばＣ言語におけるscanfやstrcmp,forやifなどの命令、prompt_readなどのマクロ命令についても、同様にステートメントの見出し表を作り仕様の作成やプログラムの生成に用いる。これらの見出し表には、目的プログラム言語で書いたモジュールの本体部を収納するファイル名項目を除き、見出しやタイプ,関数表現項目など，モジュールの見出し表と同様の項目を含んでいる。

３．モジュールの検索

仕様を作成するとき、ライブラリがどのような基本モジュールを具備しているかを調べるために、見出し表などのブラウジングによる他、見出しやキーにより属性項目を絞ったモジュールのオンライン検索を行う。このためK1,K2,...を見出しやキーとして

```
retr(K1,...,Km):-
  module(HEAD,TAIL),member(K1,L1),･･･,member(Km,Lm),
  writelist(Kn).                                          (1)
```

のような検索用の述語を適宜定義する。ここに L1,L2,...は moduleの引き数の頭部HEADか尾部TAILであって、member(Ki,Li)は Kiが HEADか TAILに含まれている項目であることを示す。そしてこの述語を用いて(2)のように、K1,K2,...,Kmの中のいくつかの項目Kr,..,Ksの値に対して残りのKa,...,Kbの属性値を適宜検索することが出きる。

```
    ?-retr(kr1,･･,kr2,Ka1,･･,Ka2)                        (2)
```

例えば、モジュールの見出しHを与えてモジュールの機能の注釈や処理対象の型を検索するには

```
head_com_type(H,COM,TYPE):-
    module(HEAD,TAIL),member(header(H),HEAD),
    member(com(COM),TAIL),member(type(TYPE),TAIL).        (3)
```

を定義し、質問

```
?-head_com_type(読み込む,X,Y)                             (2a)
```

により、入力用の各種のモジュールの日本語文による注釈説明項目や適用可能な対象データのタイプ

や構造をX,Yに逐次、検索することができる。

さて、同じ読み込み処理で、見出しが同じ”読み込む”であっても、処理対象のデータ構造や型によりＣ言語における関数つまりモジュール名や命令は異なる。しかし、始めに処理対象のデータ構造を仕様で決めた後では、簡単に同じ”読み込む”という見出しを用いても適切なモジュール名や命令が選択できるはずである。プログラム生成システムMAPPはこのような選択を処理対象のデータ構造の仕様とモジュールの見出し表を用いて、次のようなプログラムにより自動的に行う。

```
funct(PROCESS_HEADER,NAME):-                    ①
      process(PROCESS_HEADER,NAME),             ②
      obj(name(NAME),type(TYPE))                ③
      module(HEAD,TAIL),                        ④
      member(PROCESS_HEADER,HEAD),              ⑤
      member(type(TYPE),TAIL),                  ⑥
      member(funct(FUNCT,TAIL),                 ⑦
      writelist(FUNCT).                         ⑧          (4)
```

上式において①②⑤は、仕様で与えた処理の見出し項目PROCESS_HEADERをもつモジュールが検索されることを示す。②のプロセス述語や③のオブジェクト述語の引き数は仕様の述語から導出時に取り込まれ、処理の見出しとタイプの値が一致するライブラリモジュールを④⑤⑥により選択してその関数FUNCTが⑦⑧により検索され関数呼出として出力される。

このようにして、仕様で与えた各プロセスは、仕様で指定した処理対象の型やデータ構造を参照して処理対象に適合した処理関数や命令に自動的に逐次変換される。

4．仕様の整備とプログラムの作成

プログラムを作成するには先ず仕様を作成せねばならない。この節では概略仕様を、モジュールに記載したコメント文の形にオンラインで整備し、これからプログラムを自動的に生成する方法について概説する。

仕様は処理関連対象の型やデータ構造を指定するデータ構造仕様と、処理方法を指定する処理仕様に大別される。前者は、

```
objlist:-obj(x1),obj(x2),･･･,obj(xn).
```

の形で与え、xi(i=1,2,･･･)が例えば初期値をもつ配列の場合、 (5)

```
obj([name(xi),type(float),array(dim(1),size(n),max(200)),
     val([v1,v2,･･･,vn])]).                          (6)
```

のような形で与える。データ構造に関する機械可読な仕様は、上式のような単一の定型的な形でほぼ完全に与えることができる。。

次に処理の仕様であるが、これは概略仕様をユーザが日本語文などで箇条書き風に適当に作る。算術式 AL1,AL2,･･･,ALnなどは、簡単のためcomputelist(AL1,AL2,･･･,ALn)のように述語computelistの引き数部に書いておくものとする。

一方、日本語文からなる処理の仕様は短いものでも語彙の違い順序の違いなどにより、同じ意味内容の文が多くの異なる文で表されることになる。的確にユーザーの欲する処理を指定するには、その処理を行うモジュールや命令を、見出し表のコメント項目の日本語文などを参照して欲しいものを指定すればよい。

そのためには処理の中心となる事項を１～２のキーワードで指定して処理の検索を限定する。次に処理対象のデータタイプなどは無視し、この種類を行うモジュールや関数の中心機能を表す日本語文を表示させ、ユーザーの欲する処理を文番号で指定する。例えば”読み込む”などの機能を表す述語動詞や”平均”などの出力データの名詞をキーとして見出しに含むモジュールを検索し、コメント項目の日本文を表示させ、欲するモジュールのパラメータや引数を指定して作成する。

＜例１＞

問題の概要：　k=(m2-m1) 種類の商品別に、各支店の売上高を記録したk個のファイルがある。売上配列sales_tableの第0列に支店コード、第m1+1列から第m2列のk個の列に(m2-m1)種類の商品の売上を読み込み、第m-1列に各支店の総売上を求め、総売上の大きさの順にこの配列をソートし、プリントする。

問題の概要から、主な処理対象に関する次のような仕様を作る。

(1)　対象データ構造に対する仕様

```
objlist:-obj([name(sales_table),type(int),array(dim(2),size(n,m),
          max(100,8)]),
          obj([name([n,m,m1,m2]),type(int)]).
```

次に、問題の概要を箇条書きにした処理の概略仕様から、モジュールやステートメントの見出し表を用いて(1)のデータ構造に適合したモジュールを本節で概説した方法で検索し、具象化したモジュールのコメント文をコピーして、次のような機械可読の処理仕様を作る。

(2)　処理仕様

```
[[[n,m,m1,m2],をプロンプト付きで入力する],
[m1,から,m2,までの各,j,に対して],
  [sales_table.の,j,列にファイル,fn,の第,j,ファイルから列に読み込む],
[sales_table.をプリントする],
[0,から,'n-1',までの各,i,に対して],
  [j,について,m1,列から,m2,列まで,'sales_table[i][j]',を,
   'sales_table[i][m-1]',に合計する],
[sales_table,を列,'m-1',で,des,順に列ソートする],
[sales_table,をプリントする]].
```

上記の処理仕様を見出し表などを用いて目的言語に変換し、且つ、６節の手法によりメイン部頭部を作成し、付録のようなプログラムを生成する。*fnが指すファイル名や、各種モジュールの見出し表、モジュール本体のＣプログラムは別途に与えている。

５．リンクによるプログラムの部分的合成

モジュール見出し表にはモジュールの適用条件である入力項目in(IN)、処理後の状態や出力データを記述する出力項目out(OUT)を設けている。したがって、これらの知識を用いることにより、モジュールを組み合わせて作ったプログラムの実行可能性に関する論理チェックや、逆に入出力条件を与えこれを満たすプログラムを何個かのモジュールをリンクして合成することができる。ここでは後者について考え、入力データに関する 入力条件 input(IN) と typep(TYPE) なる条件の下に出力条件

output(OUT) を満たすプログラムを合成する問題を考える。このためには、次に示した出力述語output(OUT)の本体部が真となるように、本体部の述語を逐次実行すればよい。

```
output(OUT):-
    module(H,T),member(out(OUT),T),          ①
    member(in(IN),T),input(IN),              ②
    member(type(TYPE),T),typep(TYPE),        ③
    member(funct_type(FUNCTTYPE),T),         ④
    funct_declared(FUNCTTYPE),
   (member(para_obj(OBJ,type(OBJTYPE)),T),   ⑤
    para_declared_list(OBJ,OBJTYPE);none),
   (member(tmpvar_type(VARTYPE),T),          ⑥
    tmpvar_declared(VARYTYPE);none),
   (member(file_name(FILE_NAME),T),          ⑦
    included(FILE_NAME);none),
    member(funct(FUNCT),T),                  ⑧
   (state(STATE),member(FUNCT,STATE);        ⑨
    nl,writelist(FUNCT),write(';'),          ⑩
    member(com(COM),T),com_set(SET),         ⑪
    add_com_set(COM,SET),
    addstate(FUNCT,STATE)).                  ⑫                    (7)
```

まず、モジュールの尾部Tの出力項目out(OUT)の引き数が仕様の出力条件OUTと一致するモジュールを探索する。成功すればそのモジュールの入力項目in(IN)やタイプ項目の引き数TYPEが、仕様で与えた入力述語input(IN)の引き数INや　タイプ述語typep(TYPE)の引き数TYPEとそれぞれ一致するかどうかを調べる。一致すれば④～⑥で型宣言すべき関数やパラメータ変数、一時置数変数をモジュール尾部の各項目からtype_member述語の引き数部に登録し⑦でインクルードすべきファイルがあればこれを登録し、⑩でOUTを満たす手続きを与える関数表現を出力し、⑪でその関数表現に対する日本語文を注釈用にモジュール尾部のcom項目からcom_set述語の引き数部にコメント用に収録する。

もしこの様な入力データ項目Xがこのモジュールに含まれていなければ、

```
input(X):-output(X).                                              (8)
```

なる関係を用いて、Xを出力項目に含むモジュールの探索を繰り返す。探索に失敗すればプログラム作成は失敗に終わる。他方、この様な探索を繰り返して、仕様で与えられた入力条件を満たすモジュールに達すれば、これまでたどってきた探索の経路とは逆方向に始めの出力条件OUTを含むモジュールに向かって、各モジュール記載の④以降の処理、すなわち、各種の変数や関数の型の登録、ファイルの登録、関数FUNCTの出力を前述のように繰り返し、MAPPはメイン関数のプログラムを作成する。なお、登録した、変数などの型やファイルなどのデータを用いて、６節に述べる手法によりメイン関数の頭部などを作成し、プログラムを完成する。

上記の出力述語output(OUT)の他、いくつかの命令をマクロ展開的に生成する出力述語やマクロ定義を設けている。

図２は、以上のような手法により、「入力条件として、整数n,配列aのデータをキーボードより与え、大きさnの１次元配列aの配列要素の偏差を出力印字する。」というプログラムを自動生成したときの結果を示す。配列のデータを処理する数個のモジュールをリンクする他、変数nのデータをキーボードから読み込むマクロ命令を自動的にプログラムに取り込んでいる。

＜スペック＞

```
objlist:-obj(a,type(float),array(size(n)),
        obj(n,type(int)).
input_list:-input([a,n],keyboard).
proc:- output(funct_var_printed(dev([[a,n]]))),
```

＜合成されたメインプログラム＞

```
#include<c:devladm.c>
#include<c:avladm.c>
#include<c:sumladf.c>
#include<c:readladf.c>
main(){
float     dev_a,dev_lar_f(),av_a,av_lar_f(),sum_a,sum_lar_f(),a[];
int     read_lar_df(),n;
printf("n=");
scanf("%d",&n);
read_lar_df(a,n,"a");
sum_a=sum_lar_f(a,n);
av_a=av_lar_dm(sum_a,n);
dev_a=dev_lar_dm(a,n,av_a);
printf("dev_a=%f",dev_a);
}
```

＜図２＞

なお、モジュール見出し表のコメント項目を用いて、プログラムの日本語注釈文を、次に示す述語により生成することができる。その際、見出し表にあるモジュールなどの関数引数部の変数は、仕様の変数名で具象化し、置き換えている。

```
prog_com_gen([H|T]):-
        funct_com_gen(H),prog_com_gen(T).                    (9)
prog_com_gen([]).                                            (10)
funct_com_gen(FUNCT):-
        module(H,T),member(funct(FUNCT),T),
        member(com(COM),T),writelist(COM).                   (11)
```

６．メイン関数頭部の作成

前節では処理仕様から仕様の詳細化やプログラムへの変換について述べたが、Ｃプログラムの場合には、始めにインクルード関数やメイン関数などの関数記号部、型宣言部などを作成せねばならない。MAPP はこれらを処理対象に関する仕様や、検索したモジュールに含まれる情報を用いて、述語main_headにより自動的に作成する。

```
main_head:-(include([]);include(FILE),include_sent_gen(FILE)),
            main_header,type_decl.                           (12)
```

右辺の本体部はインクルード述語の引き数部が空であれば何もせずに次の述語main_headerへ行き、そうでないときには引き数部のFILEをインクルードする文をinclude_sent_gen(FILE)という述語で作

成し、その後 述語main_headerで main()などを作り、述語type_declでタイプ宣言部を作ることを示す。

プログラムで利用する、ライブラリモジュール関数を含のファイル名がまだinclude(OLD_FILE)の引き数部に含まれていなければ、次のように述語included(FILE)によりこれを述語includeの引き数部に加える。

```
included(FILE):-((include(OLD_FILE),member(FILE,OLD_FILE));
                 add_file(FILE,OLD_FILE)).                    (13)
```

mainなどの関数記号部は次の述語main_headerにより作成される。

```
main_header:-main_arg(ARG),wrielist(['main(',ARG,')']),nl,
             write(' {'),nl.                                  (14)
main_arg([]).                                                 (15)
```

main関数記号部に続く型宣言部は、処理対象におけるデータ構造の仕様obj(X)や、検索したモジュールから関連情報をtype_member述語の引き数部に抽出保持した情報に基づきtype_decl述語を用いて作成する。

まずobj述語により、引き数部に記述した型に関する情報をtype_member述語などに記録する。

```
obj(name(NAME),type(TYPE),array([dim(D),size(S),max(M)]),val(VAL)]):-
        assert(p_obj(name(NAME),type(TYPE),array([dim(D),size(S),max(M)])])),
        lang(c),type_member(TYPE_MEMBER),
        (member(NAME,MEMBER);add_type_entity([NAME,val(VAL)],TYPE,MEMBER)).
                                                              (16)
```

上式は処理対象が配列で初期値を持つ場合の、データ構造仕様の読み込みにおける処理ルールである。処理対象がスカラであったり、初期値が指定されていない場合には、arrayやvalの項目は省略する。右辺本体部のlang(c)は目的言語がＣ言語であることを指定する。

さて、処理対象の型やデータ構造の仕様は、多くの場合、見出しとして(5)のように１つのobjlist述語で与え、その本体部にいくつかのobj述語を並べて記述している。従って、obj述語から処理対象のデータ構造に関する情報を直接参照するのは不便である。このために、上式のobj述語では仕様のobjlistの本体部との単一化により処理対象のデータ構造の情報を取り込み、これをassert文でp_obj述語の引き数部に記憶し、演算や入出力命令の作成における処理対象の型名などの問い合わせに対処させている。

次のtype_member部ではこの処理対象が、そのTYPEのMEMBERに登録されているかどうかを調べ、登録されていなければ[NAME,val(VAL)]をadd_type_entity述語によりそのTYPEのMEMBERに加える。

この他、MAPPはモジュール見出し表から関数値やパラメータ変数、一時置数変数などの各種変数の名前やタイプをtype_memberに登録収納した後、次のtype_decl述語によりＣの型宣言の様式に従って印字する。

```
type_decl:- (type_member(int,[]);type_member(int,INT),
             write('int'    '),write_namelist_ic(INT),write(';'),nl).   (17)
```

上例はデータが整数型の場合のタイプ宣言部のプログラム作成部分で、この型に属するメンバが[]で変数が存在しないときには、何も印字せず、そうでないときには、引き数部の INT の変数リストを、write_namelist_ic(INT)により、要素間に','を間に挟んでwrite_name述語により印字する。

write_name述語は対象のスカラや配列の別、初期値の有無などにより異なる型宣言を出力するように設定してあり、MAPPは対象のデータ構造や初期値の有無などに従って、設定した型宣言部を出力する。

次に、write_name述語の中で、１次元配列で大きさと初期値を指定した宣言を出力するためのプログラム作成部分を示す。

```
write_name([NAME,max(M),val(VAL)):-nl,write('          '),
          writelist([NAME,'[',M,']={]),
          writelist_ic(VAL),write('}'),nl.                    (18)
```

ただしwritelist(X)は引き数Xのリスト要素を並べて印字し、writelist_ic(X)は、間に','を挟んでリスト要素を印字する。

「　付　　録　」

```
#include <c:srt_colf.c>
#include <c:sumrowgf.c>
#include <c:prt_2arf.c>
#include <c:rdcolflf.c>
main()
{
char    des;
static char  *fn[10]
float    sum_rowf();
float    sales_table[100][8];
int    i,n,m,m1,m2,j;
printf("n,m,m1,m2=");
scanf("%d%d%d%d",&n,&m,&m1,&m2);
for(j=m1;j<=m2;j++){
   rd_colflf(sales_table,n,m,j,fn,j);
}
prt_2arf(sales_table,n,m);
for(i=0;i<=n-1;i++){
   sales_table[i][m-1]=sum_rowf
                        (sales_table,i,m1,m2);
}
srt_colf(sales_table,m-1,des,n,m);
prt_2arf(sales_table,n,m);
}
```

7．あとがき

プログラムの概略を記した仕様より、モジュールの属性項目を記述したモジュールの見出し表を用いて、必要なモジュールを検索し合成するシステムをプロログで構成した。プロログを用いることにより、モジュールごとにもたせる属性項目は一律である必要がなく、したがって項目の追加等も容易である。またそのフレキシブルな記号処理能力により、仕様に日本語を用いることも容易に行うことができる。

今後は、応用上のための一層の改善を図る他、ＣＯＢＯＬ等他言語への展開・視覚的記号を用いた仕様表現の利用などについて検討していきたい。

【参　考　文　献】

[1] 西田,高松:プロログによるプログラム生成システム
情報処理学会第４２回全国大会　pp.5.241-242 (1991.3)
[2] 恐神,西田:プロログによるモジュールの検索とプログラムの合成
情報処理学会第４３回全国大会　pp.4.293-294 (1991.10)
[3] 恐神,西田:仕様の記述とプログラムの作成
情報処理学会第４４回全国大会　pp.5.191-192 (1992.3)
[4] 恐神,西田:プロログによるモジュール援用プログラミングシステム
福井工業大学研究紀要 第22号pp.4.299-306 (1992.3)

(平成４年12月18日受理)