

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

恐 神 正 博 ・ 西 田 富士夫

Module-Aided Programming System by using Prolog

Masahiro OSOGAMI and Fujio NISHIDA

As the needs for improving software productivity and quality have become hot topics, the demand for automatic program generation has increased. As for techniques concerned with program generation, various software engineering methodologies have been proposed.

This paper describes a module-aided programming system by using Prolog. Compared with other computer languages, Prolog, due to its list processing and unification techniques, is capable of powerful string processing and inference capability. These functions are very effective in constructing an automatic programming system as mentioned briefly above.

Using these features, the Module-Aided Programming system by Prolog, called MAPP, has been planned and constructed.

1. まえがき

近年、ソフトウェアの生産量の激増と複雑化にともない、いわゆるソフトウェアの危機が叫ばれて久しい。これに対して、プログラム製作過程を機械化して効率化をはかって問題を解決すべく、プログラムモジュールの再利用技術やプログラム製作の自動化などの研究が各方面で活発に行われ、かなりの成果も上げられている。筆者らもこれに関連して研究を行ってきた。今回は日本語の単語などを適宜用いた表現を許し、比較的読み易いプログラム仕様から、基本モジュールを用いてC言語などのプログラムを作成するシステムをプロログで構成する方法を考察し、MAPP(Module-Aided Programming system by Prolog)の基本システムを試作し検討したので報告する。プロログでシステムを構成する利点は

1. プロログは記号処理能力に優れているので、仕様から生成プログラムまでの各段階での表現変換を比較的簡単なプログラムで行うことができる。
2. 基本プログラムモジュールのライブラリから目的とするプログラム作成に必要ないろいろな情報や知識をフレキシブルに検索し結合することができる。
3. 単一化に基づく推論により、いくつかのモジュールを自動的にリンクしてプログラムを部分的に合成することができる。

などである。ここでは、おもに1.について述べる。

2. 仕様の記述

通常、プログラムは仕様に基づいて作成される。要求仕様についてはそれぞれの応用分野において、処理の内容に即して有用な手法が、例えば、事務処理分野におけるJackson法や制御システム設計における状態推移法など、いくつか提案されている。この報文は仕様の作成の始めに、処理関連対象の計算機内における型やデータ構造を選定してデータベースに登録したのち、これらの処理対象に、モジュールライブラリに設けられている基本的な処理ルーティンや関数プログラムを適用して要求する処理を施したり、要求するデータを出力する仕様を作成し、仕様からプログラムを半自動的に作成する手法についてその概要を述べる。

処理対象の型や構造を始めに選定するのは、入出力などの標準的な処理は、対象の型やデータ構造に応じてほぼ決まっており、仕様で指定したデータ構造に応じて既成の標準的なモジュールをカスタマイズしてを再利用するのが効率的であるからである。半ば定型的な処理は処理の主要部をモジュール

ル化して作成しておくことができ、細部はオプションにより選択したり必要に応じて小変更を加えることが容易である。また、事務処理や科学技術計算における演算などの処理も、処理対象の型やデータ構造により分類されることが多い。したがって仕様に対象の名前と、これに対する処理や演算の種類の列を2, 3の主なオプションとともに指定すると、システムは仕様の始めに与えた対象名と型やデータ構造を参照して、適用可能なライブラリモジュールを検索し、効率的に仕様の詳細化やプログラムの作成を進めることができる。

2. 1 仕様における対象の型と処理

仕様の記述は型定義、対象リスト、処理からなりプロログで

specifications:-typedefinition,objectlist,processing. (2.1)

と表すものとする。

対象Xが構造体やレコードの場合には、いろいろの型があり、型定義が必要である。これらの型を次のように宣言する。

typedefinition:-struct_type(X,memberlist([[OBJ1,TYPE1],...,[OBJn,OBJn]])). (2.2)

ここにXは型の名前であり、memberlistの引き数部は構造体などの各メンバの名前OBJiと型TYPEiの対のリストである。

対象xiの宣言は対象リスト

objlist:-object(x1,y1),...,object(xn,yn). (2.3)

としてまとめて入力する。各対象xiの型やデータ構造yiは対象がスカラの場合

object(name,type(type0)). (2.4a)

で指定する。ここにobjectは述語記号、引き数部のNAMEは対象の名前、type0は、int charなどの型を表すものとする。

対象が配列の場合には配列の大きさやインデクス変数名の項目を引き数部に加え

object(name,type(type0),array([size(m,n),index(i,j)])). (2.4b)

のように表す。

また、対象Xが構造体やレコードの場合には、(2.2)の形の型定義を用いてXの型に属する構造体配列を

object(name,struct_type(X),array(size(N),index(INDEX))). (2.4c)

と宣言する。

これらの仕様は

:?-specifications. (2.5)

により逐次読み込まれ、対象リストの各対象の名前が型などとともに、(2.6a),(2.6b)などを用いてデータベースに書き込まれる。(2.6a),(2.6b)は対象がスカラであるときの書き込みのプログラムで、インデクス変数などの対象が既にデータベースに登録されているかどうかをチェックし、登録されていなければデータベースに型とともに書き込むプログラムである。

obj(X,type(TYPE)):-member_check(X,TYPE),!. (2.6a)

obj(X,type(TYPE)):-type_member(TYPE,Y),add_entity(X,TYPE,[X Y]),
assert(type(X,TYPE)),
write(TYPE),write(' '),writelist([X Y]). (2.6b)

ただし、大文字からなる文字列は変項、小文字からなる文字列は定項を表す。上式は、Xをassert述語によりデータベースに書き込むとともに、型がTYPEのtype_memberに加え、これをディスプレイに表示することを指定する。配列や構造体の場合も同様であるが、インデクス変数の書き込みも同時に自動的に行っている。

つづいて処理の仕様においては処理の概略を表す述語表現を処理の順に記述する。

processing:- process_name1(object_name1,goal1),
.....,
process_namen(object_namen,goaln). (2.7)

ここにprocess_nameなどは処理名で引き数部に対象名や2, 3の属性値を指定するだけで、システムは仕様で与えた対象の型やデータ構造を参照して、適用可能なproceduresや関数を含むモジュールを探索し詳細化する。

2. 2 日本語仕様から形式仕様への変換

2. 1で述べた仕様は、機械による仕様の矛盾チェックや、仕様からのプログラム生成の自動化のために機械可読な形にすることが望ましい。このためには表現が明確で機械で処理し易い関数的表現が望ましいが、1バイト系の文字表現などのため、書きにくく理解しにくい。他方、日本語文などによる自然言語的表現は、人間にとっては処理し易いが、憶え易く簡明で表現力の大きい日本語文型を設定することは大きな問題で十分な検討を要する。ここでは仕様の形式的表現として、2. 1で概説したようなプロログで書いた、構造が簡単な関数的表現を選び、他方、日本語表現として関数記号や引き数部をオプションで日本語の字句に置き換えたものを基本にした機械可読な仕様を用いるものとする。また、読み易さを増すために(2.8)のように引き数には、文脈に置ける役割を示す、役割名を適宜、前置することを許すものとする。

頭部([役割名:], 対象1, ..., [役割名:], 対象n). (2.8)

ここに頭部は性質関係などの述語記号名や操作や処理などの関数記号名を表し、役割名は頭部の表す関係や処理における各対象の役割を表す。

日本語表現の"頭部(X)"を1バイト系の形式表現に機械的に変換するためには、述語記号や関数記号毎に

頭部(X) :- tran(X, XE), 引き数部の書き出し述語(head(XE)). (2.9)

のような形の変換ルールを設ける。

ここに"頭部"とこれに対応する"head"はそれぞれの述語記号や関数記号毎に与えた2バイト系及びその1バイト系表現の頭部の記号とする。また、tran(X, XE)は2バイト系の語Xを1バイト系の英字に変換したものがXEであることを示す。このような形の変換規則を与えておけば、後は引き数の変数名の2バイト系1バイト系間の単語変換辞書を設けることにより、1バイト系表現が得られる。

以下に2, 3の変換ルールを示す。

```
対象('名前:', X, '型:', 構造体, Y, '組成:', 配列([大きさ(N), インデックス(I)])) :-
    tran(X, XE), tran(Y, YE), writelist(['obj(' , XE, ', ', ' , 'struct(' , YE, ')', ' ', ' ',
    'array([size(' , N, ')', ' ', ' ', 'index(' , I, ')', ')]', ')]', nl.
```

(2.10)

```
平均('対象:', X, '出力先:', Y) :- tran_list([[X, XE], [Y, YE]]),
    gen_w([av_fc, ['obj:', XE, 'goal:', YE]]).
```

(2.11)

(2.10)は構造体配列の宣言、(2.11)は平均を求める述語の2バイト系から1バイト系表現への変換規則をそれぞれ示す。(2.15)のwritelist(X)は引き数部のXを書き出す述語である。

頭部や引き数部の各語は次のようなgen_w 述語やtranlist述語により1バイト系に変換される。仕様には始めから1バイト系の文字で書いた方が分かりやすい語もある。このような語は変換する必要がなく、始めに与えた1バイト系の字句Eはtran(E, E)によりそのまま1バイト系に渡される。

```
gen_w([PRED, ARGLIST]) :- write(PRED), write('('),
    writeqlist_ic(ARGLIST), write(')', ' '), nl.
```

(2.12)

```
tran_list([[HJ, HE] | T]) :- tran(HJ, HE), tran_list(T).
tran_list([]).
```

(2.13)

```
tran([HJ | TJ], [HE | TE]) :- tran(HJ, HE), tran(TJ, TE).
tran([], []).
```

(2.14)

```
tran(E, E).
```

```
writelst([H|T]):-write(H),writelst(T).
writelst([]).
(2.15)
```

2.3 仕様とその例

以下に販売処理に例をとって仕様の表現を示す。MAPPは(a)の日本語による概略仕様を単語辞書を援用して(b)の概略形式仕様に変換する。

(a) 日本語概略仕様

仕様:-型定義,対象リスト,処理

型:-構造体型('名前:',販売レコード,
'メンバー:',構成リスト([[支店名,文字],[コード,整数],[テレビ,実数],
[ビデオ,実数]],[ファックス,実数],
[パソコン,実数],[ワープロ,実数]])).

対象リスト:-

対象(販売リスト,'型:',構造体(販売レコード),
'組成:',配列([大きさ(n),インデックス(i)])).

対象(平均値_パソコン,型(実数)),

対象(最大値_パソコン,型(実数)).

処理:-

入力('obj:',販売リスト,'source:',キーボード), /*売上リストのキーボードからの入力*/

表示(販売リスト), /*売上リストの表示*/

av_pc=平均_出力(販売リスト,パソコン), /*パソコン売上金額の平均計算と表示*/

max_pc=最大値_出力(パソコンリスト,パソコン), /*パソコン売上金額の最大値計算と
表示*/

出力('obj:',販売リスト(i), 'cond:',ge([販売リスト(i),パソコン]),lower_limit)).

/*パソコンの売上金額がn以上の売上リストの表示*/

(b) 概略形式仕様

```
specifications:- type_definition,objectlist,processing.
type_definition:-write('type_definition:-'),
                  assert(struct_type(sales_rec,
                  complist([[ 'shop_name[20]',char],[code,integer],
                  [tv,float],[video,float],[fax,float],[pc,float],[wp,float]]))),
                  struct_def(sales_rec).
objectlist:-
    obj(sales_list,struct(sales_rec),array([size(n),index(i)])),
    obj(av_pc,type(float)),
    obj(max_pc,type(float)).
processing:-write('mainf:-'),nl,main,type_decl,func.
main:-mains,
      read_fc(sales_list),
      print_fc(sales_list),
      av_fc([sales_list,pc],av_pc),
      printf(av_pc),
      max_fc([sales_list,pc],max_pc),
      printf(max_pc),
      print_if_fc(sales_list,[ge,[sales_list,i,pc],lower_limit]),
      mains_end,nl.
```

3. 仕様の詳細化と形式仕様の生成

3. 1 詳細化における展開法と関数コール法

2節で述べた簡易仕様はライブラリモジュールの適用やユーザのオプションにより詳細化される。

仕様の手続き部において $h(t_1, t_2, \dots, t_l)$ なる処理が指定されていると、MAPPはこれと単一化可能なモジュール $h(X_1, X_2, \dots, X_l)$ をモジュールライブラリの中から探索し、これらのモジュールの中から本体部が仕様で指定した対象の型やデータ構造を満たすモジュールの詳細部を仕様に取り込む。仕様に取り込む方法には、仕様の該当部を仕様と単一化したモジュールの本体部の詳細手続きで置き換える展開法と、仕様の該当部から詳細手続きを呼び込む関数コール法とがあり、モジュールの種類によりどちらかに決めている。展開法は分岐や繰り返しなどの制御モジュールや、スカラ型データの出力など実行回数が多く手続きが短いモジュールに適用する。

関数コール法は仕様の $h(t_1, t_2, \dots, t_l)$ に対応するモジュールの関数呼出表現で置き換え、コールされる関数の本体部を、対応するモジュールの本体部の変数を仕様やオプションで指定した変数名や値に具象化(instantiate)して作る。

2. 3節の例の概略形式仕様(b)の中で、_fcを後置した関数記号が、関数コールに対応する関数である。概略仕様に_fcを含む仕様が現れたとき、MAPPはこの概略仕様から(3.1)に示すMAIN部の形式仕様の対応するコール部を作成するとともに、(3.2)に示すようなコールすべき関数モジュール群の関数名の列 `mhl(module_header list)`を生成し、次にこれら関数モジュールの形式仕様を生成して、詳細化する。

```
qis:-main,
    read_stra(sales_list),
    print_stra(sales_list),
    set_fun(av_pc,[av_stra,[sales_list,[pc]]]),
    print('obj:',av_pc,'format:',[av_pc,=,real]),
    set_fun(max_pc,[max_stra,[sales_list,[pc]]]),
    print('obj:',max_pc,'format:',[max_pc,=,real]),
    print_if_stra(sales_list),
    mainend.                                     (3.1)
```

```
mhl:-read_stra(sales_list),print_stra(sales_list),av_stra([sales_list,pc]),max_stra
    ([sales_list,pc]),print_if_stra(sales_list,[ge,[sales_list,i,pc],lower_limit]).
                                                (3.2)
```

3. 2 モジュールの構成

モジュールはCなどのコンパイラ言語で直接作成し、これをライブラリに登録して利用することが出来るが、ここでは基本的なモジュールをプロログを用いて作成しておき、これよりCやコボルなどの言語に変換しこれらの言語で書いたモジュールを作成する手法について考察する。Cなどの言語ではいろいろな型の構造体に適用可能な汎用的なモジュールを作ることは困難である。他方、以下に示すようにプロログを用いて基本的なクラスの構造体に適用可能な汎用的なモジュールを作っておき、これを仕様の構造体にカスタマイズしてC言語のモジュールをメンバの数や型に関わらず効率的に作ることができる。

一般に モジュールは、次のような規則節

```
h(X):-b1(X),b2(X),...,bn(X),
    c1(X),c2(X),...,cm(X).                     (3.3)
```

を用いて、手続きの概要を記述する。 $h(X)$ はモジュールの処理や処理後の状態を表す見出し名、

$b1(X), b2(X), \dots, bn(X)$ は処理対象の型など、モジュールに対する適用条件、

$c1(X), c2(X), \dots, cm(X)$ は $h(X)$ なる処理を行ったり、処理結果を得るための詳細を表す手続き群あるいは サブモジュール群 である。

次に構造体Xにおける、メンバCの最大値を求めるモジュールの例について述べる。

```

① max_f([X,C]):-
② p_obj(X,struct(XS,complist(L)),member([C,TYPE],L),
③ func_type_decl([[TYPE,'max_stra',[X,C]],L),
④ arg_type_decl([[struct,[XS,X]],float,C]),
⑤ begin_body,
⑥ local_type_decl([integer,I],[TYPE,temp_max]),
⑦ set(temp_max,[X,0,C]),head_for_iter(I,1,minl(M)),
⑧ maxb(temp_max,[X,I,C],temp_max),
⑨ end_body_p.

```

(3.4)

①はこのモジュールの見出し部を表す。すなわち、max_struct_array_f([X,C])は、構造体配列(struct_array)のXという処理対象のCというメンバ(属性)の最大値を求める関数呼び出し型のモジュールであることを示す。

②のp_obj(X,struct(XS,complist(L)),member([C,TYPE],L))はこのモジュールの適用条件を示す。これは、XがXSというタイプの構造体でLという構成リストをもち、CはLのメンバで、CのタイプはTYPEであることを示す。

X,CはもとよりXS,L,TYPEもすべて変数であるので、このモジュールから殆どすべての構造体配列やいくつかのレコードからなるファイルにおけるある項目の最大値を求めるプログラムモジュールが生成できることが分かる。

③から⑨までは処理の詳細部を示す。

③のfunc_type_decl([[TYPE,'max_stra',[X,C]],L)は、このモジュールから関数型のモジュールを作る場合の関数名'max_stra'を引き数部とともに示し、関数のとる値のタイプ名をTYPEで示している。

④のarg_type_decl([struct,[XS,X]],float,C)は③の関数の引き数XとCのtypeが、それぞれ構造体(struct)のXS型及び実数型(float)型であることを示す。

⑤は③の関数モジュールの本体部の始まりを示す。

⑥のlocal_type_decl([integer,I],[TYPE,temp_max])は、この関数モジュールで用いられる局所変数Iのtypeがintegerで、temp_maxのtypeがTYPEであることを示す部分である。

⑦のset(temp_max,[X,0,C]),head_for_iter(I,1,minl(M))は、temp_maxの初期値をX[0]->Cに設定し、反復文for_iter(ation)のインデックスをIとし、I=1からM-1(=minl(M))まで繰り返す制御文を生成することを指示している。

⑧のmaxb(temp_max,[X,I,C],temp_max)は、temp_maxとX[I]->Cとの中の大きい方をtemp_maxにおく関数である。

4. C への変換

4.1 仕様における述語変項や関数変項の表現

仕様の詳細化やC言語への変換を行う場合、少ない変換規則で統一的に変換や生成を行うことが望ましい。ところがPrologを含む多くのプログラミング言語において関数記号や述語記号は定項として扱わねばならない。それでMAPPでは、日本語仕様から形式仕様への変換や形式仕様からプログラミング言語への変換を統一に行うために、述語記号や関数記号を変項としこれらを含む述語表現や関数表現をリストで表し、変換処理を記述している。

さて、while やifなどで表される反復文や条件文のテスト部には任意の述語を対象として記述することが望ましい。このためにテスト述語は一般に

[PRED,T1,T2,...,Tn] (4.1)

のよう述語記号と引き数とからなるリストで表し、次のようにこれを条件述語head_if_thenや反復述語whileなどの引き数部とする。

head_if_then('cond:',[PRED,X,Y]). (4.2)

関数記号が変項である場合についても同様に、リストを用いて次式のように表す。

```
set_fun(X,[FUN,A]):-writelst([X,'=',FUN,'(']),write_term(A),
                        write(');').nl. (4.3)
```

4.2 C言語への変換

詳細化して作成した形式仕様からC言語などのプログラムを生成する。詳細化仕様からのこれらの手続き言語への変換は主として構文変換、字句変換によって行うことができる。以下に型や関数の宣言や制御文の2, 3の例について、形式仕様の関数表現をCの関数表現に変換する変換規則を示す。

```
①type_decl([TYPE, MEMBER]):- writelst([TYPE,''],
                                     type_member(TYPE, MEMBER)
                                     writelst_ic(VAR),write(';').nl.
②func_decl('type:',TYPE,'obj:',[FS,ARGLIST]):-
                                     writelst([TYPE,'',FS,'(']),
                                     writelst_ic(ARGLIST),write(')').nl.
③for('index:',I,'from:',FROM,'to:',minl(TO)):-
                                     writelst(['for(',I,',',
                                     FROM,',',I,',<=',TO,',-1',',',I,',++')']),nl.
④head_if_then('cond:',[REL,X,Y]):-
                                     p_rel(REL,PREL),
                                     write('if('),write_term(X),write(PREL),
                                     write_term(Y),write(')').nl.
④-①p_rel(le,'<=').
④-②p_rel(ge,'>=').
```

①②のtype_declaration述語は、仕様作成や詳細化の過程において(2.6b)のassert文などにより出現した関数や変数名を、TYPEの値毎に自動収集したtype_member述語のMEMBER項のリストに基づいて、これらをC言語の型宣言に書き換えている。

③④では、for文やif_then文の頭部の形式表現は、C言語の記号を用いて条件部の中置形の表現に書き換えている。一般に、関数や述語はスカラ変数、配列、構造体、関数などいろいろな種類の項を引き数とする。これに対応して述語変項や関数変項をリストで表して処理するMAPの仕様表現においては、いろいろの種類の項を表1のような表記を用いて表している。これらの形式表現の項は⑤以下のwrite_term述語により、組織的にC言語に変換する。

たとえば仕様でのif_thenの条件部は、式④の変換規則の頭部のように表しこれよりCのif_then文の条件部を生成している。そしてXやYの項(term)は仕様に指示するTERMの種類に応じて、write_term関数によりC言語表現に変換する。

項の種類	仕様における表記	Cにおける表記
スカラ変数	T	T
1次元配列変数	[T1,T2]	T1[T2]
2次元配列変数	[T1,[T2,T3]]	T1[T2][T3]
構造体	[T1,T2,T3]	T1[T2]->T3
構造体	[T1,[T3]]	T1->T3

表1

```
⑤write_term([X,I,C]):-write(X),write_bra_list(I),writelst(['->',C]).
⑥write_term([X,[C]]):-write(X),writelst(['->',C]).
```

```

⑦ write_term([X,I]):-write(X),write_bra_list(I).
⑧ write_term(X):-write(X).
⑨ write_bra_list([H T]):-write(' '),write(H),write(' '),write_bra_list(T).
⑩ write_bra_list([]).
⑪ write_bra_list(I):-write(' '),write(I),write(' ').

```

5. あとがき

基本的なライブラリモジュールを用いて、日本語単語の使用などを許す機械可読のプログラム仕様からC言語プログラムを作成するシステムを考察しプロログで構成した。プロログはリスプや単一化などの記号処理能力をもち、上記システムの基本部分は比較的容易に構築することが出来た。今後は応用上のいろいろな立場からの検討改善をはかる他、モジュールの検索、リンク、他のプログラム言語への展開などを検討していきたい。

【 参 考 文 献 】

- 1) 西田, 高松: プロログによるプログラム自動生成システム, 第42回情報処理学会全国大会 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自動合成, 第43回情報処理学会全国大会, pp.4-293~294(1991.10).
- 3) S.Takamatsu,T.Nishida: Transformation between informal Expressions and Formal Expressions in Program Specifications, Trans. of IEICE, vol.E73, No.5, pp.729-737(1990.5).

付録

2. 3節の仕様からMAPPによるC言語のプログラム生成ならびに生成されたプログラムの実行などの実験を行なった。付録Aはメインプログラム, 付録Bは最大値を求めるプログラムのモジュールの部分である。

付録 A

```

struct sales_rec{
char    shop_name[20];
int     code;
float   tv;
float   video;
float   fax;
float   pc;
float   wp;
}
main(){
struct sales_rec sales_list[20];
float max_stra(),av_stra(),max_pc,av_pc;
int i,n;

printf("n=");
scanf("%d",&n);
read_stra(sales_list);
print_stra(sales_list);
av_pc=av_stra(sales_list,"pc",n);

```

```

printf("av_pc=%f\n",av_pc);
max_pc=max_stra(sales_list,"pc",n);
printf("max_pc=%f\n",max_pc);
print_if_stra(sales_list);
}

```

付録 B

```

float max_stra(sales_list,pc,n)
struct sales_rec sales_list[];
char pc;
int n;
{
int i;
float t_max;
t_max=sales_list[0].pc;
for(i=1;i<=n-1;i++);
if(t_max<=sales_list[i].pc)
t_max=sales_list[i].pc;
return(t_max);
}

```

(平成3年12月19日受理)