プログラム生成におけるモジュールの自動リンク

恐神正博* ・ 西田富士夫**

Automatic Module Linking in Program Generation

Masahiro OSOGAMI ・ Fujio NISHIDA

Many automatic program_generation systems lack the ability to adequately describe the capability of various aspects of program components. They can also difficulties in describing program specifications and design.

This paper presents a new approach for module_linking in automatic program generation. Input and Output conditions, data_types used for variables and procedures for each module are separately defined in a predicate form. They are then combined into a rule_clause form of Prolog used to define the function of a module. Using this method increases efficiency and flexibility of Program specifications and construction.

1. まえがき

プログラムとは入出力条件などからなる情報処理の要求に基づき、処理単位の命令や関数モジュールを組み合わせて作成した手続きであり、プログラミングとは与えられた情報処理の要求に対して既知の処理単位をつないで要求の処理を実現する問題解決の手続きである。

このような観点から従来、プログラムの自動合成の研究や検証などの基礎的研究が行われてきた。近年ではメニュー方式とGUIの利用により、プログラムの生産性はかなり向上している。しかしながら、モジュールの検索や変形、リンクなどについては、いまだ大きく人手に依存しているようであり、これらの自動化が望まれている。自動合成の研究、とりわけモジュール間の自動リンクの研究が余り進展しないのは、モジュールやシステムの機能や条件を簡明且つ弾力的に表現する方法が確立されておらず、仕様や設計を手軽に記述し処理することが困難な事情によることがあげられる。

この論文はモジュールの入力条件あるいは適用条件、出力条件あるいは機能、モジュールに含まれる変数のデータタイプなどの事項を別々に述語で表し[1][2]、これらをモジュール毎に規則節に組み合わせて表すことによりモジュールのリンクや設計の詳細化などの処理を簡明且つ柔軟に行う一つの方法を考察し検討したものである。

2. 入出力条件

処理の要求仕様は入力条件と出力条件とに分かれる。入力条件とは(1)(2)のように処理に必要な対象のデータ構造などの属性に関するデータタイプ条件やデータの存在場所や前処理などに関する条件である。

---

*事務局　**経営工学科

```
dt(n,[var,int]).                                    ①
dt(a,[ar(200),int]).                                ②
                                                    (1)

in([in_keyboard,n]).                                ①
in([in_seq_file,fl2,b]).                            ②
in([in_mem,i])).                                    ③
                                                    (2)
```

ここに(1)の述語 dt(X, T) はXのデータタイプはTであることを示す。例えば①はnが整数型変数であることを表し、②はaが最大200の要素からなる整数型の1次元配列であることを表す。(2)の述語記号 in は入力条件を表し、引数部はデータの存在場所や先行処理の結果の状態やデータ名などを表す。(2)の①②③はそれぞれデータ n, b, i がキーボード、ファイル fl2、メモリに与えられていることを表す。これらは Prolog において事実を述べる事実節である。第一引数部は概ね処理や状態、関係の名前を表し、第二以下の引数部は入力や出力などの変数名を表す。

プログラムにより処理してえられる結果の状態や出力されるデータは出力条件として、述語や関数形式にまとめる。そしてこれを出力述語の引数として out 記号をつけて表す。

入力条件の下に、n個の指定した出力条件 out(X1), out(X2), . . . , out(Xn) を満たすプログラムを求めるには、出力述語を用いて、背理法を変形した Prolog の質問節の形

```
?|-out(X1),out(X2),...,out(Xn).                     (3)
```

で表す。例えば [a, の(配列要素の)和を, t_sum, に求める], [t_sum, を印字する] と指定するには次のような質問節の形で表す。

```
?:-out([sum,a,t_sum]),out([print,t_sum]).           (3a)
```

3. 処理の記述とリンク

前節で述べたように問題の入力条件を与え、指定した出力条件を満たすプログラムを求めるためには、いくつかの処理モジュールを自動的にリンクすることが必要である。このために各目的言語の基本命令やプロシジャーをモジュール化しその適用条件あるいは入力条件や、機能あるいは出力条件を機械可読の形で明らかにしておかねばならない。これは Prolog の規則節を用いて式(4)のような形で比較的容易に定式化することができる。小文字だけからなる文字列は定項を表し、大文字を含む記号列は代入などができる変項を表す。

```
out(OUT):-dt(D,T),in(IN),proc(PROC),
          add_type_entity_list(TYPE_LIST).          (4)
```

式(4)は頭部の述語が成立して OUT の処理が出力されたり、状態が成立するには、本体部が成立すればよいことを表す。すなわち、入出力の変数Dのタイプが T であり、手続き PROC の適用条件あるいは入力条件が IN であるとき、手続き PROC を行えばよく、プログラム作成においては PROC の呼出表現を書き出せばよいことを表す。

手続き PROC には展開形と呼出形とがある。展開形は基本関数やこれらを数個

組み合わせて複合機能をもたせたものでマクロ展開と同様に、プログラム生成のときにout述語の呼出しもとに、これらの基本関数のタイプや変数名などをカスタマイズしたものをそのまま書き出す。C言語では例えばout述語による呼出が数個のデータの読み込みや出力の場合、scanf, printなどの基本関数を書き出す。out述語がプロンプト付き読み込みの場合にはPROC部はprintf("入力フロンプト文")とscanfなどの関数からなる。選択や反復の制御機能の呼び出しにおいてもその制御部分のコードの展開出力には(9)で述べるように簡単のためこの方法によっている。手続が呼出形の場合には、proc述語の処理はこの処理呼出の呼出元への書き出しと、カスタマイズした処理本体ソースコードのfunct_list述語の引数部への取り込みからなる。

add_type_entity_list述語はその引数部に含まれる変数が新しく出現したものであれば、タイプ別にtype_member述語の引数リストに加えて宣言部の作成に利用する。

以下例により説明する。(5)は一次元配列Xの要素の和sumをTSUMに求める処理を記述する規則節である。

```
out([sum,X,TSUM]):-
    dt(X,[ar(N),TYPE]),in([in_mem,X]),                    ①
    proc([sum,[[ar(N),TYPE],X],TSUM]),                    ②
    add_type_entity_list([[X,TYPE],[N,int],[T_SUM,TYPE]]). ③
```
(5)

上式は出力述語が成立するためには、①②③が成立するか実行すればよいことを示す。①は入力条件を示し、Xが大きさNでTYPEがint, long, doubleなどに指定された一次元配列であり、そのデータが、例えば初期設定、読み込み、計算などにより、メモリの中にすでに取り込まれてin_memが成立することを示す。②のproc述語は処理を書き出す部分である。一般にproc述語は処理毎に詳細化のための定義や手続きを(6)のように目的言語のコードを発生するまで階層的に与える。

```
proc([sum,[[ar(N),TYPE],X],TSUM]):-
set_to(suml a(X,N),TSUM),add_funct(suml a(TYPE)).          (6)
```

(6)の本体部はまずset_to述語により、T_SUM=suml a(X, N); の形のカスタマイズ表現をメインプログラムなどの呼出もとに書き出す。続いてadd_funct述語により、タイプTYPEをカスタマイズした関数suml a(TYPE)の関数定義の本体をfunct_list述語の引数部に取り込む。③においては、この処理においてタイプ宣言が必要な変数名とタイプをタイプリストに加え記憶する。タイプリストへの登録においては既に登録されているか二重定義はないかなどのチェックを行う。

ここで各処理においてモジュールの適用条件が成立することが必要であり、次の規則により処理する。

```
in(Z):-pi(PI),member(Z,PI).    ①
in(Z):-out(Z),add_state(Z).    ②
```
(7)

このモジュールを適用しようとするプログラム合成問題の入力条件をpi(PI)とする。①はモジュールの適用条件Zが成立するためにはZがPIの元として含まれていることが保証されていればよいことを示す。②はPrologで書いたこのプログラム合成プログラムにおいて①の後に記述され、①が成立せずこのままでは入

力条件が成立しないときにアクセスして適用される。②は入力条件Zを成立させるためにZを出力述語とし問題の入力条件PIを満たす手続の列が求まるまで再帰的に探索することを示す。add_state述語は出力述語Zの重複探索をさけるためにZを記録し探索のとき参照する。

データXにある処理を施す場合その処理の入力条件はデータXがメモリ上にあることである。メモリ上にないときには、各処理のモジュールの規則節毎に入力条件として、Xをキーボードからプロンプト表示付きで読む、ファイルFLからﾎﾟｲﾝﾀfprを用いて読むなどと記載しておかねばならない。このため、一般のデータのロードに関する出力述語と入力述語の間の規則として次式を設ける。

```
out([in_mem,X]):-in([prompt_read,X]).                    ①
out([in_mem,X]):-in([read_seqfile,XDATA,X]).             ②
```

(8)

式(8)の本体部の各入力述語in(P)に対応してその出力述語out(P)に関する規則節を設ける。それらの規則節においてprompt_readやread_seqfileの出力述語の入力条件は(2)の①②である。

なお、これらの出力述語名は簡単のため処理対象のスカラ、配列、構造体のデータ構造にわたり同じ名前を用いている。しかし処理対象のデータタイプdt(D, T)により、出力を保証するproc(PROC)の内容が異なるので、おなじ出力述語を持ちデータタイプやproc(PROC)などの本体部が異なる規則節がいくつも存在する。従って出力条件を満たすモジュール列を探索するとき、処理対象のデータタイプdt(D, T)に適合した規則節を選択する。

仕様において(1)(2)のような入力条件のもとに(3)のような出力条件を与えると、実験システムMAPPは仕様の出力条件と単一化可能な出力述語を頭部にもつ規則節を探索し入力条件を調べる。与えられた仕様の入力条件と単一化できれば、proc述語により手続き呼出関数が出力される。そうでなければこの規則節の適用条件を満たすための前処理が(7)により再帰的に探索され、(2)のような仕様の入力条件によりモジュールの入力条件が保証されるまで実行される。探索に成功すれば前処理すなわち処理の呼出関数が各出力述語のproc述語により正順に書き出され手続き部のプログラムが求められる。

このような処理は順次処理だけではなく、選択、繰り返しなどの制御処理にも適用することができる。例えば、引数ARGにテスト条件Tが成立する場合や繰り返しのカウンタ条件が与えられたとき、処理を行ってOUTの状態になることを表す規則節は次のように表すことができる。

```
out([if(test([T,ARG])),OUT]):-
    in([in_mem,ARG]),
    if(test([T,ARG])),begin,out(OUT),end.                ①
out([if_else(test([T,ARG])),OUT1,OUT2]):-
    in([in_mem,ARG]),
    if(test([T,ARG])),begin,out(OUT1),end,
    begin,out(OUT2),end.                                 ②
```

```
out([while(test([T,ARG])),OUT]):-
    in([in_mem,ARG]),
    while(test([T,ARG])),begin,out(OUT),end.                    ③
out([forall([I,from(M),to(N),step(1)]),
out(PROC)]):-
    dt([I,M,N],[var,int]),greater_than(N,M),
    in([in_mem,M]),in([in_mem,N]),
    add_state([in_mem,I]),
    repeat_head(I,[M,N,1]),begin,out(PROC),
    end.                                                        ④
```

(9)

制御の条件が複雑でない限り、選択や繰り返しの制御部は、上式からif述語やrepeat_head述語により直接に目的言語で書き出すことができる。制御条件の一つの形式であるテスト条件にはproc述語と同様に、真偽の出力に関する手続きが比較的簡単に指定できる展開形と複雑な呼出形がある。呼出型は真偽のテストそのものが選択や繰り返しの処理を含み展開形では記述が困難な条件である。この場合には、真偽決定の手続きや定義をcall_test述語として作成しておき、これを次の規則を設けて呼び出す。

```
if(call_test([T,ARG])):-
    write_head(if,head(T,ARG)),add_funct(call_test([T.ARG])).   (10)
```

すなわち、write_head述語によりif(T(ARG))の形のテスト条件文を目的言語で呼出元に書き出す他、add_funct述語によりテストTの定義本体を関数定義リストに加える。while述語のテスト条件の場合も同様である。

次に、"mからn(ただしn>m)までの正整数iのうちiが素数のものを出力する。" という問題について考えると、ライブラリの出力述語の日本語見出しを参照して、入力条件は[[i, m, n]は整数型である]、出力条件は[[i=m,から,n,までのすべての,i,に対して], [[もし,prime(i),なら], [i,をプリントする]]]という表現に置き換えられる。

その述語表現は見出し辞書より

入力条件：

```
    dt([i,m,n],[var,int]).                                      (11)
```

出力条件：

```
?:-out([forall([i,from(m),to(n),step(1)]),
        out([if(call_test([prime,i])),[print,i] ]) ]).          (12)
```

が自動的に求まり、(c)に対する計算機出力は

```
for(i=m;i<=n;++1){
    if(prime(i)){printf("%8d",i);}                              (13)
```

となり、prime(i)の定義本体はfile_list(FL)の引数部FLに

```
int prime(int x){ int i;
        for(i=2;i<x;i++) {if(x%i==0) return 0;}
        return 1; }                                             (14)
```

として記憶される。

4．プログラムの生成

リンク手法などの手法により自動的に作成した手続きのコード部に、必要な情報を付加し、書式を整えてコンパイル可能なプログラムを作成する。式(15)はCの書式に従って全プログラムを作成する手順を示す。

```
prog(q(N)):-
    include,funct_def,main_head,q(N),end_main.          (15)
```

Nは作成要求のプログラム番号を示す。include述語はout述語において必要に応じてinclude_list述語の引数部に取り込むように指示したファイル名の一覧を

```
include:-
    (include_list([]);include_list(L),write_include_list(L)).    (16)
```

により書き出す。ここにinclude_list([])は、事実節のこの述語の内容が空リストであれば何も書き出さないことを指定する。funct_def述語もfunct_list述語の引数部に作成した呼出関数の定義部をinclude述語と同様の方法で書き出す。

main_head述語はメイン頭部、メイン部で出現する関数や変数の宣言などを書き出す。このうち変数の宣言部の作成は次のように行う。モジュールのout述語などにおいて宣言が必要な変数はadd_type_entity述語の引数部に指定され、この述語が呼び出される毎にタイプ毎にtype_member述語の引数部に記憶される。type_member述語の引き数部の内容はmain_headの本体部のtype_decl述語

```
type_decl:-
    (type_member(char,[]);type_member(char,CHAR),
       write_type_member('char      ',CHAR,';'),nl,
    (type_member(int,[]); ..............................),nl.    (17)
```

により、書き出して宣言部を作る。q(n)はメインの手続き部を、end_mainは'}'を書き出す。

以上はメイン関数の作成について述べたが、メイン頭部を関数定義の頭部表現におきかえることにより通常の関数モジュールを作成することができる。

5．日本語文による処理

仕様における入出力条件や関数呼出文は、これまで、2，3節のようにout述語などの述語表現を用いて表してきたが、読み書きがしにくく、日本語文などを使用できることが望ましい。仕様の入出力条件は例えば処理の種類毎に集めた汎用処理モジュールのメニューの一覧表から、メニュー方式で処理の見出しを選定し、組み合わせ、引数値を指定して作成する。この見出しの内容はout述語などに対応するが、見易い日本語文や表形式の表現が望ましい。このため見出しの日本語文表現などと述語表現との間に(14)のような変換辞書を設ける。辞書の各jp項目の引数のうち、第1要素は項目番号、第2要素は日本語文表現、第3要素は述語表現である。

処理メニューの一覧表から、見出しの日本語表現の番号と、引数の値を指定していくつかの入力条件文や出力条件文を作る。これらの条件文は、1文毎に(19)のconv述語の引数部Xに取り込み、変換辞書を参照して述語形に変換してファイルに書き出す。

```
conv_dict([
  jp([11,[Xは,大きさ,N,最大値,M,の一次元配列でﾀｲﾌﾟは,TYPE,である],
  dt(X,[ar(N),TYPE])]),
  jp([111,[ﾃﾞｰﾀ,X,はkbから与える],in([in_kb,X])]),
  jp([555,[X,の和を,TSUM,に求める],out([sum,X,TSUM])]),..]).        (18)
conv(X):-conv_dict(D),member(jp(X,Y),D),
        write(out(Y)),nl.                                          (19)
```

語述語形の入力条件、出力条件は、メモリに読み込んでデータベースに加え、これより質問形の out 述語を実行して所要のプログラムを生成する。

生成されたプログラムには、理解を容易にするため手続き呼出表現に対応して日本語文などのコメント文を付加することが望ましい。ここでは簡単のため手続き呼出の日本語表現をその out 述語の日本語表現で代用することとする。コメント文を付記するために各 out(X) 述語の規則節の本体部に add_com(X) を設け、(16) により、out 述語の手続き呼出部をプログラムに取り込む毎に、out(X) に対応する日本語表現を (18) の変換辞書により求め、com_list 述語の引数部に取り込み、オプションにより引数部の内容をコメントとしてまとめて書き出す。

```
add_com(X):-com_list(CL),conv_dict(CD),
        member(jp(J,X),CD),append(CL,[J],NCL),
        retract(com_list(CL)),assert(com_list(NCL)).               (20)
```

6. 例題

"ファイル fl の n 個の実数型データ x(x の下限は lb, 上限は ub からクラス数 class_n の度数分布配列 freq(class_n) を作り、棒グラフに表示する。"という問題を考える。出力述語見出し表を参照して、出力条件、入力条件を見出し表にあるものから選び変数名を指定して書き換える。

入力条件

```
[x,は大きさ,n,のデータで整数型である]                              ①
[x,はシーケンシャルファイル,fl,の中にある]                         ②
[[n,lb,ub,class_num],はキーボードから与える]                       ③
```

出力条件

```
[データ,x,を分類して頻度表,freq_cnt,を作る],                       ④
[クラス数,class_num,の頻度表,freq_cnt,を棒グラフで表す]            ⑤
```

(21)

(21) ～ (25) の日本語文表現からその述語表現は変換辞書と (19) から次のようになる。

入力条件

```
dt(x,[size(n),int]).                                               ①
in([in_seqfile,fl,x]).                                             ②
in([in_keyboard,[n,lb,ub,class_num,]).                             ③
```

出力条件

```
out([classify,x,freq_cnt]),                                    ④
out([bar_graph,x,freq_cnt]).                                   ⑤
```

(22)

仕様の出力条件を表すout述語のclassifyとbar_graphに関する入出力規則は次のように与えられている。ただし本体部のadd_type_entity_listなどの述語は簡単のため・・・.と表して省略している。

```
out([bar_graph,X,N]):-in([classify,X,FREQ_CNT]),
    dt(FREQ_CNT,[ar(N),TYPE]),
    in([max,FREQ_CNT,TMAX]),in(in_mem,X),
    proc(bar_graph,[FREQ_CNT,TMAX]),・・・.                    ①
out([classify,X,FREQ_CNT]):-
    dt(X,[ar(N),TYPE]),dt(FREQ_CNT,[ar(CLASS_NUM),int]),
    dt([N,LB,UB,CLASS_NUM],int),
    in([in_mem,[X,N,LB,UB,CLASS_NUM]),
    proc(classify,[X,N,LB,UB,CLASS_NUM]),・・・.                ②
```

(23)

(31)から(33)のようなout([max, freq_cnt, TMAX])の述語を通じて，freq_cnt配列の最大値をTMAXに求めることが必要である。また、(32)と(8)からは、out([in_mem, x])から(33)のout(read_seqfile(fl, x)を通じて、xをファイルから配列の形で読み込み、n, lb, ub, class_num　をキーボードから読み込むことが必要であることがわかる。実験システムMAPPはこれらの関係と補足情報を用い、基本モジュールをカスタマイズしリンクして全プログログラムを出力した。

```
out([max,X,TMAX]):-dt(X,[ar(N),TYPE]),in([in_mem,X]),
    proc(max,[ar(N),TYPE,X],TMAX),・・・.                      ①
out([read_seqfile,FILE_NAME,X]):-
    dt(X,[size(N),TYPE]),in([in_seqfile(FILE_NAME,X),
    dt(X,[array(N),TYPE]),in([in_mem,N]),
    proc(read_seqfl,[FILE_NAME,X,N]),・・・.                    ②
```

(24)

7．あとがき

モジュールの出力条件、入力条件、データタイプ、手続き部などを別々に述語形式で表し、これらを規則節の形で結びつけることにより、モジュールのリンク、選定などの処理の記述が簡潔になり、より弾力的になった。今後は仕様や設計が概略から詳細へ移行できるように詳細指定機能が付加できることが望ましい。

【参考文献】

[1] 恐神正博，西田富士夫：プロログによるモジュールの検索とプログラムの合成，福井工業大学研究紀要　第23号，pp. 313-320(1993)

[2] F. Nishida et al:Semi-Automatic Program Construction From Specifications Using Library Modules, IEEE Trans., SE, vol. 17, no. 9, pp. 853-870, Sept., 1991.

(平成8年12月17日受理)