入出力条件を用いたプログラムの自動生成

恐　神　正　博*

Automatic Program Generation using Input and Output Conditions

Masahiro Osogami

Abstract

Since the 1980's, object oriented programming and structured programming have been hot topics for increasing software productivity. Usually, software has been handmade while other products are machine made. However, demand for software has been increasing due to the influence of factory and office automation, so a software crisis has begun.

In this paper, experiment results of automatic program generation using MAPP (Module Aided Programming system by Prolog) are reported. MAPP makes C language programs from easy specifications which includes natural language. Also during program generation, MAPP tries to compensate for any missing specifications which were not completely described using input and output conditions of modules.

１．まえがき

１９８０年代の中ごろから構造化プログラミング設計法やオブジェクト指向の手法等が提唱され，プログラミングの生産性向上の研究及びそれらの実用化が進められている．一方，従来より手作業によって行われてきたプログラミングという作業を機械化する試み，すなわちプログラムの自動生成に関する技術についても研究が進められている．

ここでは昨年度報告した手法に基づき [6]，実験システム MAPP の開発・改良を進め，プログラムの概略設計文からそれらの入出力条件を用い，設計文において欠落していた条件を自動的に補填しながら，Ｃ言語のプログラムを自動生成する実験の結果を報告する．

＊経営工学科

２．プログラム自動生成の現状

１９４５年にアメリカのペンシルバニア大学において世界初の電子計算機 ENIAC が開発されて以来，電子計算機はわずか数十年の間にめざましい発展を遂げ，現在，産業構造の根幹を支える重要な役割を担うまでに至っている．一方，人間の要求を手続きとして機械へ翻訳するいわゆるプログラミングという作業は，要求が大きければ大きいほど繁雑になり，また，専門的な知識・技術を必要とする特殊な分野という位置付けから脱却しきれていない．そのため，多くの手工業的技術はロボット等の機械に取って代わられてきたにもかかわらず，ことプログラミングに関してはいまだに職人芸的な手作業によって，その供給がまかなわれ続けている．しかしながらソフトウェア（いわゆるプログラム）の需要は年々増加・複雑化の一途を辿っており，ソフトウェアの生産が，量・質共，その需要に追いつかないという，危機的な状況が現出しようとしている．このことは，１９６８年ドイツのガルシュミッツで開催されたＮＡＴＯの国際会議で既に指摘されており，それ以来，各方面においてソフトウェアの生産性を向上させるために，自然言語によるプログラミング，モジュールの再利用化，プログラミングの自動化など様々な研究が行われて来ている．しかしながら，いまだに決定的な解決策は見つかっておらず，各方面において研究が続けられているのが現状である．

ここでは昨年度報告した手法に基づき [6]，実験システム MAPP の開発・改良を進め，プログラムの概略設計文からモジュールの入出力条件を用い，設計文において欠落していた条件を自動的に補填しながら，Ｃ言語のプログラムを自動生成する実験の結果を報告し，その有用性について検証を行う．

３．ソースコードの生成[6]

昨年度の紀要で報告した手法により，概略設計文の準備及びＣのソースコードの生成について簡単に述べる．

３．１　概略設計文の作成

プログラムを設計仕様に基づき，Prolog のリスト形式を用い概略設計文の作成を行う [4]．概略設計文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．データ構造仕様は，

obj_list(spec(NUM)):-obj(x1), obj(x2),…,obj(xn).　　(3-1)

の形で与えられる．

次に処理仕様であるが，これは概略設計文を利用者が自然言語を用いた表現で行い

MAPPの書式に合わせ記述する．一方，自然言語からなる処理の仕様は短いものでも語彙・順序の違いなどにより，同じ意味内容を多くの異なる文で表すことができる．このためMAPPにおいては，処理の中心となる事項をキーワードで指定して処理の検索を行い，要求に見合う処理手続き文を選択し，それに必要なカスタマイズを施しながら設計文に付け加えていく方法をとることで，的確なMAPPの書式に合わせ解決している．

ここで処理仕様は，それぞれの処理項目をprocess_1,process_2,…,process_nとした時，

```
process(spec(NUM)):-
    proc_list([ process_1, process_2, … , process_n ]).          (3-2)
```

で与えられる．ここでprocess_nは自然言語による表現が用いられている．また，この時のspec(NUM)は式(3-1)で与えられるスペック番号に対応しており，それぞれの処理項目に対応してモジュール辞書が与えられている．[6]

3．2．ソースコードの生成

式(3-1),(3-2)がそろった後，以下の述語式(3-3)を実行することでCのソースコードの生成が行われるが，詳細については前回の報告と重複するため省略する．

```
prog(spec(NUM)):-
    lang(c),obj_list(spec(NUM)),
    main_head,process(spec(NUM)),
    main_end.                                                    (3-3)
```

4．入出力条件のチェック

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条件の項目in(IN)，処理後の状態や出力データを記述する出力条件の項目out(OUT)を設けている．従って，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いくつかのモジュールをリンクして生成することができる．ここでは後者に付いて考え，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生成する問題を考える．このためには次に示した出力述語ouput(OUT)の本体部が真となるように，本体部の述語を逐次実行していけばよい．

```
output(OUT):-
   module(Head,Tail), member(out(OUT),Tail),                    ①
   member(in(IN),Tail), input(IN),                              ②
   member(type(TYPE),Tail), typep(TYPE),                        ③
   memebr(funct_type(FUNCTTYPE), Tail),                         ④
   funct_declared(FUNCTTYPE),
   (memebr(para_obj(OBJ,type(OBJTYPE)), Tail),                  ⑤
   para_declared_list(OBJ,OBJTYPE);none),
   (member(tmpvar_type(VARTYPE), Tail),                         ⑥
    tmpvar_declared(VARTYPE);none),
   (member(file_name(FILE_NAME), Tail),                         ⑦
   include(FILE_NAME);name), member(funct(FUNCT),Tail),         ⑧
   (state(STATE), member(FUNCT,STATE);                          ⑨
       nl,writelist(FUNCT), write(','),                         ⑩
   member(com(COM),Tail), com_set(SET),                         ⑪
   add_com_set(COM,SET), addstate(FUNCT,STATE)).                ⑫        (4-1)
```

まず，モジュールの尾部 Tail 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，一時置換変数をモジュール尾部の各項目から type_member 述語の引き数に登録し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 術語の引き数部にコメント用として収録する．

もし，このような入力データが含まれていなければ，

```
input(X):-output(X).                                            (4-2)
```

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，このよう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初めの出力条件 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．

以上の概念図を図１，図２に示す．

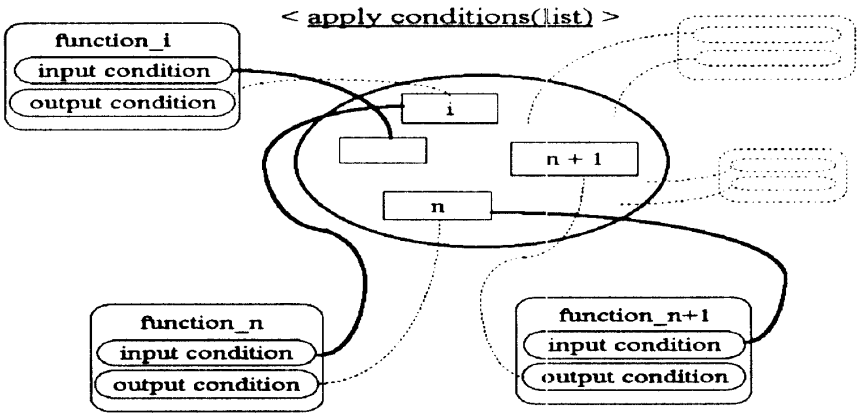

図１．リスト処理を用いた入出力条件のチェック

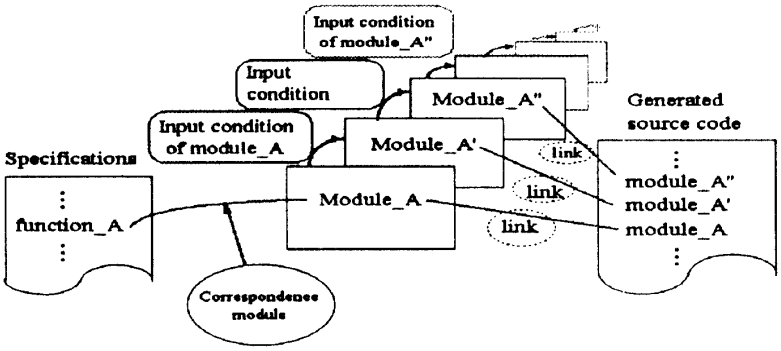

図２．設計文における欠落部分の自動補填

図１については，プログラム生成時に各モジュールの入出力条件を蓄えるリスト(Apply conditions list) を準備しておき，リンクされるモジュールについての入力条件がリスト上に既に存在しているかどうかをチェックし，リンク可能であればリンクされたモジュールについての出力条件を，新たにリスト上に追加していくことを表している．

図２については，図１のようにチェックされた段階で入力条件が満たされていなかった場合に，今度はMAPPが登録されているモジュール群から必要とされている入力条件を持つ新たなモジュールを検索し，設計文に指定されていなくとも，生成されるプログラムに追加していくことを表している．

５．実験及び考察

５．１　実験

前述の手法に基づき，以下の例題について実験を行った．

例題：大きさｎの１次元配列ａにキーボードよりデータを入力し，それらの配列要素の標準偏差，和，平均を出力印字する．その際，整数ｎ，配列ａは入力条件として与えている．

この場合，和，平均，標準偏差の出力印字を組み合わせを替えていくつか行い，その結果出力されるソースコードに違いが出るかどうかを確認する．

上の例題について，処理仕様におけるそれぞれの組み合わせを替えて考える．和，平均，標準偏差の順序で処理を与えたものを実験１，標準偏差，和，平均で処理を与えたものを実験２，和のみで処理を与えたものを実験３，標準偏差のみで処理を与えたものを実験４とし，それぞれについて自動生成を行いそれらの結果を比べる．

ここで，全ての問題におけるデータ構造仕様は図３のようになる．

```
objlist:-obj(a,type(float),
         array(size(n)),
         obj(n,type(int)).
```

図３　全実験におけるデータ構造仕様

また，実験１，実験２，実験３，実験４の結果はそれぞれ図４，図５，図６，図７のようになる．

５．２　考察

すべての実験について，変数の読み込みについては設計文に指定されていないにもかかわらず，生成されたプログラムにはMAPPによって変数の読み込み部分が自動的に補填されていることがわかる．また，実験２については求める結果の順序を入れ替えているが，出力される結果の順序が入れ替わっているだけで，プログラム内部の計算順序は同じであることがわかる．これは，実験２において標準偏差を求める段階で各変数の合計，平均が必要となってくるためで，標準偏差のモジュールがリンクされた段階でその入力条件である和，平均を求めるモジュールが既にリンクされてしまうためである．この後，設計文の指示どおり和及び平均のモジュールがリンクされようとするが，すでにリンク済であるためMAPPによってリンクの重複が避けられていることがわかる．実験４も設計文においては標準偏差のみの指定しかないが，生成されたプログラムは実験１，実験２とほぼ同じ

であり，出力結果において標準偏差のみとなっていることがわかる．

```
spec:-
 output(funct_var_printed(sum([[a,n]]))),
 output(funct_var_printed(av([[a,n]]))).
 output(funct_var_printed(dev([[a,n]]))),

#include     "sum1adf.c"
#include     "dev1adm.c"
#include     "av1adm.c"
#include     "read1adf.c"
main(){
  float     sum_a,sum_1ar_f(),
            av_a,av_1ar_f(),
            dev_a,dev_1ar_f(),
            a[];
  int       read_1ar_df(),n;
  printf("n=");
  scanf("%d", &n);
  read_1ar_df(a,n,"a");
  sum_a=sum_1ar_f(a,n);
  av_a=av_1ar_dm(sum_a,n);
  dev_a=dev_1ar_dm(a,n,av_a);
  printf("sum_a=%f", sum_a);
  printf("av_a=%f", av_a);
  printf("dev_a=%f", dev_a);
}
```

図４．実験１における処理仕様
及び自動生成された
Ｃのソースコード

```
spec:-
 output(funct_var_printed(dev([[a,n]]))),
 output(funct_var_printed(sum([[a,n]]))),
 output(funct_var_printed(av([[a,n]]))).

#include     "dev1adm.c"
#include     "av1adm.c"
#include     "sum1adf.c"
#include     "read1adf.c"
main(){
  float    dev_a,dev_1ar_f(),
           av_a,av_1ar_f(),
           sum_a,sum_1ar_f(),
           a[];
  int       read_1ar_df(),n;
  printf("n=");
  scanf("%d", &n);
  read_1ar_df(a,n,"a");
  sum_a=sum_1ar_f(a,n);
  av_a=av_1ar_dm(sum_a,n);
  dev_a=dev_1ar_dm(a,n,av_a);
  printf("dev_a=%f", dev_a);
  printf("sum_a=%f", sum_a);
  printf("av_a=%f", av_a);
}
```

図５．実験２における処理仕様
及び自動生成された
Ｃのソースコード

```
spec:-
 output(funct_var_printed(sum([[a,n]]))),

#include     "sum1adf.c"
#include     "read1adf.c"
main(){
  float    sum_a,sum_1ar_f(),
           a[];
  int       read_1ar_df(),n;
  printf("n=");
  scanf("%d", &n);
  read_1ar_df(a,n,"a");
  sum_a=sum_1ar_f(a,n);
  printf("sum_a=%f", sum_a);
}
```

図６．実験３における処理仕様
及び自動生成された
Ｃのソースコード

```
spec:-
 output(funct_var_printed(dev([[a,n]]))),

#include     "dev1adm.c"
#include     "av1adm.c"
#include     "sum1adf.c"
#include     "read1adf.c"
main(){
  float    dev_a,dev_1ar_f(),
           av_a,av_1ar_f(),
           sum_a,sum_1ar_f(),
           a[];
  int       read_1ar_df(),n;
  printf("n=");
  scanf("%d", &n);
  read_1ar_df(a,n,"a");
  sum_a=sum_1ar_f(a,n);
  av_a=av_1ar_dm(sum_a,n);
  dev_a=dev_1ar_dm(a,n,av_a);
  printf("dev_a=%f", dev_a);
  printf("sum_a=%f", sum_a);
  printf("av_a=%f", av_a);
}
```

図７．実験４における処理仕様
及び自動生成された
Ｃのソースコード

６．あとがき

汎用的なモジュールライブラリを用意しておき，それらについての入出力条件を持たせておくことで，簡単な概略設計文より必要なモジュールを検索，リンク，あるいは設計段階で不足していたモジュール等の自動補填を行いつつ，Ｃ言語のメインプログラムを生成する方法について実験を通し検証を行った．

今回の実験では，最終的に必要な結果のみを設計文に与えてやることで，残りの部分についてはMAPPが自動的に補填しつつプログラムを自動生成できることを示した．

実際にシステムを構築する場合，様々な要素が複雑に絡み合うため，実験のように簡単な指定のみでプログラムを生成しシステムを構築するのは現実的には難しい．しかしながら，自動補填という考え方は，ソフトウェア自身によるプログラムの自動生成を考える際の，一つの有用な手段になりうることを示すことができた．

今回は順次処理についてのみの実験を行ったが，今後は，より複雑なシステムに対応すべく，制御構造・反復処理を含んだ条件設定及びその検証法が課題である．

◇ 参 考 文 献 ◇

[1] 原田　実： CASEのすべて，オーム社，1991.

[2] 西田，高松：プロログによるプログラム自動生成システム，
情報処理学会第４２回全国大会，pp.5-241-242(1991.10)

[3] 恐神，西田：入出力条件によるプログラムの合成，
情報処理学会第５２回全国大会，pp.5-47-48(1996.3)

[4] 恐神，西田：概略設計文からのＣソースコードの生成，
福井工業大学研究紀要第２５号，pp.315-322(1995.3).

[5] M. Osogami, F. Nishida: "A Method of Automatic Program Designing and Code Generation using Informal Procedure Call Sentences",
Proc. of IASTED - ASC'98, pp.161-164, May 1998(Mexico).

[6] 恐神：プログラム生成におけるリスト処理を用いた入出力条件のチェック，
福井工業大学研究紀要第２９号，pp.289-296(1999.3).

[7] M. Osogami: "A Method of Automatic Program Generation and Stractured Diagram using Informal Procedure Call Sentences",
Proc. of IASTED - SEA'99, pp.104-108, Oct. 1999(Arizona).

（平成11年12月１日受理）