プログラム設計と構造化図作成の自動化

恐神正博*・西田富士夫**

A method of Automated Program Design and A Structured Program Diagram Drawing

Masahiro Osogami and Fujio Nishida

Abstract

Recently, the automated design of programs has been actively developed basaed on the concept of program reuse for improving software productivity.

This paper presents a promising method of automated program design and a structured program diagram drawing which has been fabricated in our experimental system, MAPP(now under construction). Users retrieve call sentences for statements or program modules written with readable expressions(as in their native languages), then edit and customize them to meet specifications. A method for the automatic drawing of a structured program diagram for inspecting the program under construction is also presented.

1．はじめに

プログラムの生産性を上げるために、近年、国内外においてプログラム設計の自動化についての研究が活発に行なわれている。[1] この研究はプログラムで用いられる命令や基本モジュールの呼出文を、日本語その他ユーザに分かりやすく紛れが少ない表層表現とし、所望のキーに属するものをディスプレイに表示させる。さらに、視覚により適切なものを選びカスタマイズしたものを集めて設計文書を作り、これを目的のプログラム言語に一意的に変換することにより、プログラム作成を自動化しようとするものである。尚、作成中の設計文書を検討しながら設計を進めて行くために、設計文を見易い構造化図に順次自動化する一つの手法について述べる。

2．見出し文検索と設計文書作成の基本的な考え方

プログラミングの自動化の有力な一つの方法は既成のプログラムを、あるまとまり毎に基本単位に分け、一般化して収集しておき、これを目的に応じカスタマイズし、新しく与えられた仕様に合うようにプログラムを合成しようとする方法である。ここでは工業製品の標準部品と同じく、プログラムの標準部品を作っておき、これを能率的に検索する方法について考える。

プログラムの基本要素は、命令文や基本関数などでここではステートメントと呼ぶ。やや大きいものでは、多くの場合、いくつかのステートメントを組み合わせてユーザーが作ったもの

＊電気工学科(大学院生)　＊＊経営工学科

であり、ここではモジュールと呼ぶこととする。これらの呼出文は機械可読にするため目的プログラミング言語そのものか、ある文法にしたがって取り扱われることが多い。このため算術式やテスト条件式を除いて読みにくく、さらに表記法は一義的で文法的規制があるので、正しく表記するためには習熟が必要である。例として

「１０進数データをキーボードから，X，に読み込む」　(1)

という命令を考える。C言語では

scanf("%d, &X);　(2)

という命令がある。この呼出文はキー、readに属しreadを入力すればこれに属するステートメントや配列や構造体読み込みのモジュールの呼出文がともに表示される。ユーザは希望する呼出文を視察により選択し、呼出文番号などにより指定する。したがって呼出文の言語や文型は応用分野などにしたがってかなり自由に選択することができる。

(1)のXはカスタマイズすべき変数でユーザがキーボードなどから指定する。(2)の%dのタイプ記号や&記号は５節の処理対象のデータ設計で指定したものが自動的に命令コードに充当される。

3　設計文書の作成

3．1　概要

プログラム設計文は、prog([H1, H2, ...]). のように日本語文や数式などからなるいくつかの呼出文H1, H2, ... をリスト要素として与え、各 リスト要素は命令やモジュールの呼出文に対応して、ステートメントまたはモジュールの引き数として処理される。

prog([H|T]):-(statement(H); module(H)), prog(T).
prog([]).　(3)

しかし、ユーザが呼び出し文をある文法の下に始めて作成して入力するのであればこれらの呼出文は、ある一定の書式や文法に従っていなければならず、文法を修得して設計文書を作成することは、少なからずユーザーの負担を増すことになる。

そこで、機能を大別しいくつかの憶え易いキーワードを手がかりに目的の機能をもつ命令やモジュールを既成の呼出文から検索し、対話形式で必要なカスタマイズ情報をパラメータとして呼出文にはめ込んで、手続き仕様を作成する方法をとる。既成の呼出文になければ新しくモジュールをステートメントなどから作成する。

3．2　命令・モジュールの検索と設計文書の作成

それぞれ大別された機能ごとに、日本語や英語のキーワードを設け、キーワードを入力してこのキーワードをもつステートメントやモジュールの一覧を表示させ適用したいものを番号で指定して検索する。

これらの辞書の書式を(4)に、実例を図１に示す。(4)の各 J_KEYi, E_KEYi (i=1, 2, ・・・) はそれぞれ日本語，英語によるキーワードを表す。英語のキーワードは入力し易いなどの点で便利

である。body部のpの引き数の中、nは呼出文の機能選択のための番号を表わす。

```
dict1([
  [head(J_KEY 1, E_KEY 1), body([p(1, V11, S11]), ..., p([n, V1n, S1n)]),
      [head(J_KEY 2, E_KEY 2), body([p(2, V21, S21], ..... p([n, V2n、S2n)]),
      ..................................................])  ]).  (4)
```

Siはこのモジュールや命令の日本語文などによる呼出文、ViはSiに含まれるカスマタイズ用の変数リストである。呼出文はリスト形などで書き、カスタマイズ用の変数の前後を図1に示すようにコンマで区切っている。

```
dict1([head(書き出す, print),
 body([p(1, X, [X, をプリントする]),
       p(2, X, [1次元配列, X, をプリントする]),
       p(3, X, [2次元配列, X, をプリントする]),..
                (a)非制御呼出文の場合
dict2([head(もし, if),
 body([p(1, T, if(T), c([if(T), then, b(S), end_if])),
       p(2, T, if_else(T), c([if(T), then, b(S1), else, b(S2), end_if])),
       p(3, T, OBJ, when(OBJ), c([when(OBJ), b(SP), end_when_list])),
       p(4, T, INDEX_LABEL), case(INDEX_LABEL),
                c([case(INDEX_LABEL), b(SP), end_case_list])),...
                (b)制御呼出文の場合
```

図1　見出し辞書の一部

```
add_sp(X):-
       (retr(X, CUR_SP), out_sp(CUR_SP);
        retrc(X, XC), spc(XC, CUR_SP))
       sp_all(SP_ALL),
       append(SP_ALL, [CUR_SP], N_SP_ALL),
       retract(sp_all(SP_ALL)),
       assert(sp_all(N_SP_ALL)),...                                    (5)
```

(5)は所要の機能の種類の呼出文を検索して処理設計文に加える述語を示す。すなわち、所要の処理のカテゴリをキーXを引き数として述語 add_sp(X)を入力すると、Xが非制御関連のキーか制御関係の手続きのキーかによって、retr述語かretrc述語が作動して図1の dict1、dict2のような辞書が表示され検索する。

非制御モジュールや命令の呼出文の場合、retr(X, CUR_SP)述語によりキーXに属する呼出文一覧を表示させ、キーボードから入力した番号の呼出文を取り出した後、ユーザがプログラムで用いる変数名リストを入力すると希望する呼出文が作成される。

次にout_sp述語により、カスタマイズした呼出文を画面に表示確認の後、sp_all述語の引き数のこれまで作成した設計文へアペンドする。

図1(b)のdict2(X)に制御関係の命令文やモジュールの呼出文の一部を示す。図1と比べて第4引き数を新しく設けている。制御の呼出文は分岐と繰り返しともに条件部と本体部からなる。条件部の式は単なる引き数とは異なり、一般には複雑な構造をもつが、文字列としては有限長で確定しておりプログラム設計時にdict1の場合と同様に第2引き数の変数指定部にまとめて指定することができる。しかし制御文の処理本体部の構造は制御文をネスト状に含むなど設計の初期の段階では、多様で長さも不定である。そこで設計の初めの段階では制御文の種類の番号と変数として条件式だけを指定し、本体部は構造と引き数部を逐次トップダウン的に指定していく方法をとる。すなわち、retrc述語により条件部の指定を終わると、続い引き続き「本体部の指定をして下さい」というspc(XC, CLR_SP)のプロンプトにより、ユーザは処理部の指定を行う。本体部は例えばif文が1個、if_else文が2個、case文が2個以上というように制御文の種類により異なるので、制御文の種類毎にspc述語を設けている。しかしどのspc述語も本体部は各枝毎に処理の最上位のキーリスト[K1, K2, ..., Kn]の指定を要求する。ユーザはたとえば[if, read, print]を本体部処理のキーとして入力する。spc述語は最初のキーK1に属する命令やモジュールの呼出文の表示を行い、呼出文を指定する。呼出文が制御文の場合には必要に応じて第2レベルの設計設定に移る。K1に対する設計指定が終わるとキー K2に対する設計に移る。

設計の進行にともない第4引き数部の見出し文の諸パラメータは指定されたものから逐次埋められていく。なお、第4引き数部の先頭のcはcの引き数部が制御文であること、bは引き数部が制御文の本体部であることを構造化図作成ルーチンやプログラムへの変換ルーチンに示す記号であり、ユーザは構造明示のための括弧表記などに煩わされることなく引き数（列）だけをシステムからの要求にしてがって与えて行けばよい。

なお、変数sp_all(X)の引き数Xに記憶した処理設計文書は、(6)に示すように述語sp_recの引き数部に、ファイル名FNと番号NUMをつけて、:?-のプロンプトに続き、reg_sp(NUM, FN).の述語を入力し、適宜、ファイルに記録する。

```
reg_sp(NUM, FN):-tell a(FN), sp_all(X), writelist(['sp_rec(', NUM, ',']),
                writeq(X), writeq(').'), told.                    (6)
```

4. 構造化図の作成 [2),3)]

プログラムの構造は構造化図の方が構造化文より見易く、このため構造化図によるプログラムの作成や表示が各方面で研究開発されている。この節では構造化のための制御ステートメントをif, thenなどのタイプ情報を用いて見出し文にレベル線や制御マークをつけて構造化図を自動的に描き、視覚性を増加する1つの手法について述べる。

構造化図には視覚性をよくするために機能やタイプ別にいろいろの形の枠を用い、この中に文字情報を押し込めるものもあるが、文字情報の長さの制限が強いので、ＳＰＤ図に準拠して制御

の種類の概要は特殊記号を組み合わせて表し、文字情報は横線の上側や右端の所定の位置におく手法をとっている。

各プログラム単位はレベル0のステートメントstartとendで自動的にはさむ。呼出文は前の呼出文から、縦線分を指定した個数だけ隔てて下方に書くが、仕様文の文頭に制御を表す語（if, then, else, while, for など）が現れた場合には、次の仕様文のレベルを1レベル増し、文頭を1レベル右に移動して書く。

呼出文の集合である設計文書のリストは述語pic_sp述語の引き数部に与えられ、各呼出文は(7)のようにstatement_p述語により処理される。

```
pic_sp([H|T]):-statement_p(H), pic_sp(T).
pic_sp([]).                                                    (7)
```

処理Pが非制御処理の場合には、(8)の①に示すように、これ以前にthen, else, caseなどの語が出て、Pを書く接続方向を示す述語がcon(right)であるときには、Pを右方向に書き次に接続方向述語をcon(down)として次に書く方向を下方向に設定する。接続方向が右方向でなく下方向であった場合には、②に示すように、write_level_line述語により第0レベルから現在の第Lレベルまで L + 1 個のレベル線を下方向に2単位伸ばした後、(9)のarg述語により第Lレベルの右側にPをかく。

```
statement_p(P):-level(L), space(IS),
   (con(right), arg(P), retract(con(right)), assert(con(down));      ①
    write_level_line(L), nl, write_level_line(L), arg(P)).           ②   (8)
arg(P):-writelist(['—:', P]), nl.                                        (9)
```

また、制御文に関しては、そのテスト条件部などの図的表現のためにそれぞれ述語を設ける。(10)はif文の場合で、write_symbol述語で第1引き数に示す'－'記号を第2引き数に示す回数の3回分、繰り返し書き、レベルを1だけ増してlevel述語の引き数に記憶し、その右に分岐を表す記号'<>'と条件文 T を書く。

```
arg(if(T)):-
        write_symbol('－', 3),
        level(L), L1 is L+1, retract(level(L)), assert(level(L1)),
        writelist(['<>—(IF:', T, ')']), nl.                          (10)
```

if文の場合、thenやelseによりさらにレベルが1レベル増すので、end_ifが現れたときには、レベルを2レベル減少させる。繰り返し文の場合も同様で繰り返しを表す記号として○などの記号を文頭に置き、右端に繰り返しの種類と条件を文字列で表して簡単であるが視認性の向上を図っている。

構造化図はディスプレイ上にXウィンドウズや通常の単一画面に適宜表示させ、これまでに作成した設計文書の構造を視覚的に検討しながら作成を進めることができる。図2にこのような方法で作成した構造化図を示す。

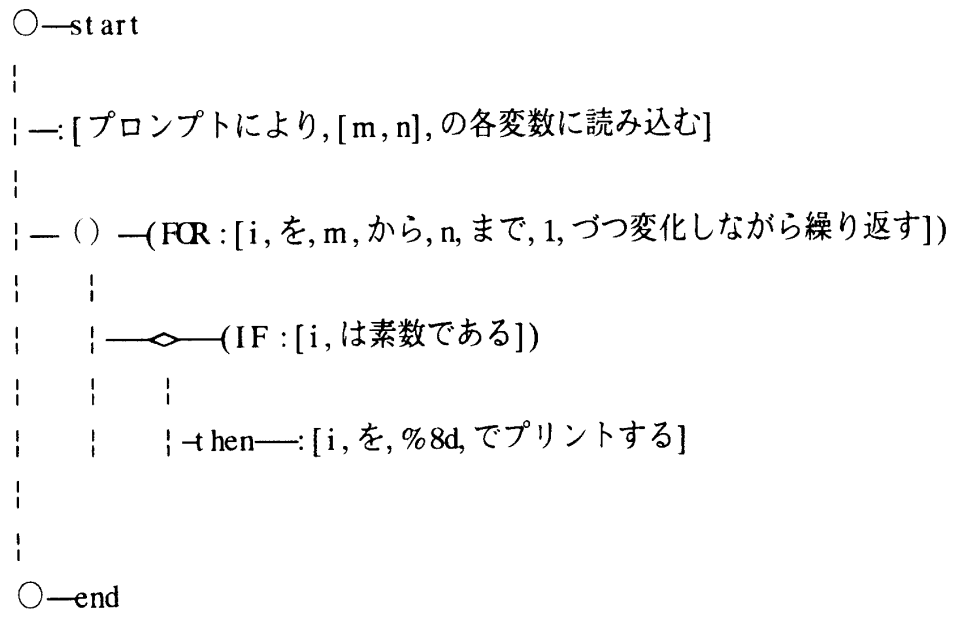

図2　構造化図の例

ファイルFNの設計文書のspec(NUM)の内容を構造化図により図示するにはプロンプトに続き、draw_pic_sp(NUM,FN).を入力する。

```
draw_pic_sp(NUM,FN):-[FN],sp_rec(NUM,X),tell('pic.pl'),
    writelist([pic(NUM),':-pic_sp([start,']),expand_list(X),
    write('end ]).'),told,['pic.pl'],pic(NUM).                    (11)
```

上式はファイルFNのsp_rec述語の引き数部の処理設計文リストXをexpand_list述語で整形処理した後、その前後にstartとendをつけ、ファイルpic.plに書き込み、これを読み出して、pic(NUM)のタグにより(7)のpic_sp述語の引き数部を構造化図して図示するものである。

5. データ構造とタイプの設計

プログラムの生成は処理対象をプログラム設計文書に従って、処理手続きを詳細化し目的言語のプログラムに変換する。従って前節までの処理設計の前に処理に現れる主要な対象のデータ構

番号	名前	タイプ・配列	初期値	備考
1	NAME	TYPE	VAL	REM
2	NAMELIST	TYPE		REM
3	NAME	TYPE	VAL	REM
		dim(D)	size(SIZE)	m_size(M_SIZE)
4	TYPE	struct(TYPE)		REM
	NAME1	TYPE1		REM1
	...	...	...	...
	NAMEn	TYPEn		REMn

図3

造やタイプの選定、必要に応じて初期値の付与などのデータ設計を行わねばならない。
図3に処理対象主要なデータ構造やタイプを指定したり表示する表を示す。

1は単一変数、2は同じタイプをもつ複数の変数、3は配列、4は構造体について書式を示す。REMは必要に応じて仮名漢字表記、使用目的種別などを書く。
TYPEはint, float, char, stringなどの他、構造体の場合には4の構造体のタイプ名を書く。3の配列ではDは配列の次元、SIZEは処理における配列の大きさ、
M_SIZEは処理の初めに確保する配列の最大の大きさ、4は構造体の定義でTYPEは構造体のタイプ名、NAME1, ・・・, NAMEnはメンバの名前を表す。

ユーザはデータ設計を行うときには図のような表をディスプレイに表示させ所望のデータタイプを番号で指定すると、MAPPは指定すべきデータの値と形を表示する。図3の番号(1)や(4)の例では

[NAME, TYPE, VAL, REM]. ①

[TYPE, REM, [[NAME1, TYPE1, REM1], ・・・, [NAMEn, TYPEn, REMn]]. ④ (12)

である。ユーザはこれらの指示に従ってリスト形式で値を指定する。無指定の場合には空リスト[]や空白表記' 'で表示する。

このようにして入力したデータ設計は、設計文書のプログラム言語への展開時に適用するステートメントやモジュールの処理対象のデータ構造やタイプに関する整合性のチェックや、変数のタイプ宣言の作成に参照する。このとき図3のような表の形式のままでは取扱いにくい。したがって、データ設計をセーブするには、(12)の①④などのデータから(14)のような述語data_spec_formを用いて、図3のようなデータ設計表を(12)のようなプロログの規則形に変換しファイルなどに記録する。

objlist(spec(NUM)):-
obj([name(NAME1), type(TYPE1), val(VAL1), ・・・]),
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・
obj([name(NAMEn), type(TYPEn), val(VALn), ・・・]). (13)

ここにspec(NUM)は仕様や問題の番号でobj(X)は処理対象やその関連対象の諸属性を記述する述語である。

data_spec_form([
[num(1), [NAME, TYPE, REM], [NAME, NN, TYPE, NT, REN, NR],
obj([name(NAME, NN), type(TYPE, NT), rem(REM, NR)])],
・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・
[num(4), [TYPE, REM, NAME_TYPE_REM_LIST],
[TYPE, NT, REM, NR, NAME_NN_TYPE_NT_REM_NR_LIST],
obj([name(TYPE, NT), type(TYPE, NT), rem(REM, NR),
struct_mem_list(NAME_NN_TYPE_NT_REM_NR_LIST)])]]). (14)

data_spec_form述語では４個の引き数部に分かれ、各引き数部は番号、(12)に示す入力属性値部、入力属性値＋字数部、述語形からなる。パタンマッチの述語を定義して、①④のような入力属性値から(13)のような指定した述語形に変換する。しかし設計途中や保守時などにおいてデータ構造を見るには、(13)のような述語形では一般にはやや見にくく、図３のような表形式が望ましい。このときには各属性値表記の開始列が揃うことが望ましい。ＴＡＢ情報が利用できないような場合には各属性値の文字数情報が必要である。このためＭＡＰＰではNAME, TYPEなどの各入力属性値の文字数NN, NTなどをカウントしこれを各引き数部の第３引き数部と第４引き数部に自動的に記録する。

このように述語形でファイルなどに記録されたデータを、適宜，オプションにより図３のような表形式に(15)(16)などを用いて変換して表示する。

```
disp(MDI, N):-[MDI], obj_list(spec(N)).                          (15)
obj([name(NAME, NN), type(TYPE, NT), val(VAL, NV), rem(REM, NR)]):-
        write(NAME), tab(15-NN), write(TYPE), tab(30-15-NT),
        write(VAL), tab(45-30-NV), write(REM), nl.                (16)
```

ここにMDIは spec(N)のデータ設計の(13)のような述語形を収納しているファイル名、tab(X)はXバイトだけスペースを空ける述語である。

6. おわりに

ステートメントやモジュールの呼出文のグループをキーによりディスプレイに表示し所要のものを選択しカスタマイズし、対応する構造化図をXウィンドウズ上に展開し、それを確認しながら設計を進める手法について述べた。仕様書や上流過程の概略設計の下、具体的に実現可能なモジュールやステートメントの呼出文を処理の種別毎に一覧参照し、カスタマイズすることにより、誤りの少ない分かりやすい設計を効率的に進めることができる。

以上のように作成した５節のデータ設計と３，４節の処理設計から、目的言語への変換辞書などを用いて、自動的にＣ言語やＣＯＢＯＬなどのプログラムを生成することができるが、報告は次回にゆずる。

【　参　考　文　献　】

1）竹下亨：ソフトウェアの保守・再開発と再利用，共立出版，１９９２

2）甲斐義久他：構造化プログラム設計図法，共立出版，１９９２

3）恐神，西田：仕様からの構造化図の簡易作成とプログラムの合成，第４６回情報処理学会全国大会，pp. 5-327～328(1993. 3)

4）恐神，西田：プロログによるモジュール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. 1-313～320(1993. 3)

(平成５年12月18日受理)