ライブラリモジュールのリンクの手法による仕様の詳細化と誤りの検出†

西田富士夫†† 藤田米春†† 高松忍††

本論文は，モジュール間のリンクの手法を用いて，ライブラリモジュールを結合して仕様を詳細化したり，仕様の誤りを検出する一つの方法について述べている．ライブラリモジュールの主要部は見出し部とオペレーション部よりなる．見出し部はモジュールを検索するのに用い，オペレーション部は見出し部に対するより詳細な手続きを記述する．使用者は MAPS とよぶ詳細化システムにプログラム仕様の専門分野の名前を入力し，その分野のモジュールのリストを提示させる．使用者はモジュールの見出し部を参照して仕様を作成し調整して，システムに入力する．MAPS は，仕様を走査し入出力表現で仕様を規定しているブロックを見いだすと，入出力条件を満たすモジュールの見出し列を Prolog により見つけ，仕様と単一化したモジュールのオペレーション部のリンクでブロックを置き換えて仕様を詳細化する．仕様のブロックが繰返しのような制御表現を含んでいる場合には，MAPS は問題を本体部のリンクの問題に簡単化し，詳細化した本体部を制御フレームに埋め込んで処理を行う．仕様の誤りは仕様の各ブロック間での接続条件の不適合によることが多い．MAPS ではライブラリモジュールを用いて仕様の書き誤りや脱落などの誤りをチェックするほか，入力された仕様をいったん入出力表現に変換し入力条件などの誤りをブロックごとに検討する．

1. まえがき

近年，ソフトウェア作成の増加とともに，ソフトウェア生産の合理化や半自動化を目指して仕様の形式化，プログラムのモジュール化，文書化などに関する研究が各方面で進められている[1)~3)]．

一般に多くのプログラムは，それぞれの分野の処理に関する知識に基づいて，基本的なモジュールを組み合わせて構成できるものと考えられる．例えば，事務処理分野では，表を生成，削除，更新したり，与えられた条件を満たす要素を探索するいくつかの基本的なモジュールがあれば，これに処理対象分野の具体的な知識を加えることにより，多くの処理を行うことができる．

筆者らはこのような観点から，基本的で共通的なライブラリモジュール（以後モジュールとよぶ）を一般化し，これを与えられたプログラム仕様に適用してプログラムを作成する手法について検討した[4)]．ここでは与えられた仕様に対し適用可能なモジュールを半自動的に連結（以後リンクと呼ぶ）して仕様を詳細化する手法と，与えられた仕様の誤りを検出する手法を中心に考察し，実験システム MAPS (library-Module Aided Program Synthesizing system) を構成して実験した結果を報告する．

† Refinement and Error Detection of Program Specifications by a Linking Technique of Library Modules by FUJIO NISHIDA, YONEHARU FUJITA and SHINOBU TAKAMATSU (Department of Electrical Engineering, College of Engineering, University of Osaka Prefecture).

†† 大阪府立大学工学部電気工学科

始めに，モジュールのリンクや仕様のチェックを行うため，仕様やモジュールの入出力述語表現などについて述べる．次にこれらの入出力述語表現を用いて，仕様と部分的に単一化したモジュールの手続き部を直並列にリンクし，仕様を満たすプログラムを合成する方法について述べる．モジュールの縦続形のリンクについては Prolog などを用いて一般的に行う方法について述べ，分岐や繰返しについては標準的なフレームに変換した後，制御部と本体部に分けて制御のフレームごとにリンクを行う方法について述べる．つづいて，ユーザの与えた仕様が，変数や入力データなどに関して所定の条件を満たしているかどうか，ライブラリモジュールを用いてリンクの手法により論理的にチェックする方法について述べる．

2. 仕様とモジュール

プログラムの仕様やライブラリモジュールは，入出力関係を規定する述語表現や，手続的表現や制御表現などを用いて記述する．これらの表現は前の論文[4)]で述べたように，いずれも，述語名や手続名，制御文名の後に，引数の役割名とタームの対の列をつけたものであり，その一つのまとまりをブロックとよぶ．役割名として，*SOurce*（処理対象を含むデータの集まり，以下 *SO* と略記する），*OBJ*（処理対象），*COND*（処理条件），*PARTIC*（*OBJ* が表す対象を処理する時に必要な補助データ），*GOal*（処理結果の格納場所，*GO* と略記する），*LOC*（データが与えられる場所）などを設ける．これらの役割名はユーザに表現の構造を想

起しやすくさせ，また自然言語でこれらの表現の注釈を自動的に生成する[8]ときなどに有用であるが，自明なときには簡単のため省略することがある．

この章ではモジュールの見出し部やユーザの仕様に用いられるこれらの表現法について説明する．

2.1 入出力述語表現

入出力述語表現は次のようないくつかの述語の連言よりなる入力表現と，いくつかのホーンクローズの連言よりなる出力表現の対からなる．

$$IN: GIVEN(OBJ: \boldsymbol{x}, LOC: \boldsymbol{u}), P(OBJ: \boldsymbol{x}) \quad (1)$$

$$OUT: Q(OBJ: z, PARTIC: \boldsymbol{x}) \quad (2)$$

上式は入力変数 $\boldsymbol{x}$ の値が $\boldsymbol{u}$ に与えられかつ $\boldsymbol{x}$ が P なる性質をもつとき，出力変数 z が入力変数 $\boldsymbol{x}$ に対して関係 Q をもつことを表す．また，式(2)の OUT 部は，$\forall \boldsymbol{x} \exists z\, Q(z, \boldsymbol{x})$ に対するスコーレム関数 $q(\boldsymbol{x})$ を用いて

$$OUT: GIVEN(OBJ: q(\boldsymbol{x}), LOC: z) \quad (3)$$

と表すことができる．入力述語と同じ述語記号 *GIVEN* を用いたこの出力表現は3.1節で述べるようにモジュールのリンクに便利な表現である．

以下，小文字からなる記号列は変項を表し大文字からなる記号列は定項を表す．仕様に現れる変数は，モジュールとの単一化 (*unification*) において定項と見なし，大文字とする．また，データや変数の型は *TYPE* 部に書くものとする．入出力表現は，次に述べる手続的表現とともに，モジュールの見出し部やユーザの仕様の記述に用いる．

2.2 手続的表現と広義の関数表現

手続的表現の主要部は

$$手続名\ (OBJ: t_1, COND: t_2, \cdots, GO: t_n) \quad (4)$$

のように操作を表す手続名，処理対象 t_1 を示す *OBJ* 部，処理対象 t_1 の満たすべき条件 t_2 を示す *COND* 部，結果を格納する変数 t_n を示す *GOal* 部などよりなる．(4)において，*GOal* 部をとり除いた表現は演算結果そのものを表すものと考え，広義の関数を表すものとする．

2.3 一般的な制御表現

仕様にはコンパイラ言語における条件分岐や反復文のような一般的な制御表現が必要である．ここでは，それらの表現について述べる．ただし以下では *GIVEN* を簡単に G と表す．また，制御表現における *OBJ* 格は，*COND* 格などの制御条件のもとで実行される処理やその結果を表すものとする．

〔1〕 条件分岐型

条件分岐の基本形は入力データ $\boldsymbol{x}$ がテスト述語 $t(\boldsymbol{x})$ を満たすとき出力条件 $G(q(\boldsymbol{x}))$ が指定されている形であり，手続表現部，入出力表現部を次のように表す．

$$\begin{aligned} &PROC: IF\_THEN(COND: t(\boldsymbol{x}),\\ &\qquad OBJ: q(\boldsymbol{x}), GO: z),\\ &IN: G(\boldsymbol{x}),\\ &OUT: \neg\, t(\boldsymbol{x}) \vee G(OBJ: q(\boldsymbol{x}), LOC: z), \end{aligned} \quad (5)$$

同様に，入力データ $\boldsymbol{x}$ が互いに排他的なテスト述語 $t_i(\boldsymbol{x})$ $(i=1,2,\cdots,n)$ ごとに出力条件 $G(q_i(\boldsymbol{x}))$ を満たすように指定されているときには，その手続表現部ならびに入出力表現部を次のように表す．

$$\begin{aligned} &PROC: CASE(COND1: t_1(\boldsymbol{x}), OBJ1: q_1(\boldsymbol{x}),\\ &\qquad \cdots\cdots\cdots\cdots\\ &\qquad CONDn: t_n(\boldsymbol{x}), OBJn: q_n(\boldsymbol{x}),\\ &\qquad GO: z),\\ &IN: G(\boldsymbol{x}),\\ &OUT: \neg\, t_1(\boldsymbol{x}) \vee G(OBJ: q_1(\boldsymbol{x}), LOC: z)\\ &\qquad \cdots\cdots\cdots\cdots\\ &\qquad \neg\, t_n(\boldsymbol{x}) \vee G(OBJ: q_n(\boldsymbol{x}), LOC: z), \end{aligned} \quad (6)$$

上式は(5)の分岐の基本形を n 個並べたものに等価である．なお，$n=2$ のときには手続き名 *CASE* は *IF_THEN_ELSE* に置き換えられる．

〔2〕 繰返し型

繰返し型の仕様は並列型と直列型に大別される．

〔2.1〕 並列繰返し型

これはベクトルの和などの計算を表すもので，スカラ量の演算などの手続的表現を用いてその演算を並列的に繰り返す場合であり次のように表す．

$$\begin{aligned} &PROC: FOR(INDEX: i, FROM: m, TO: n,\\ &\qquad OBJ: proc(OBJ: x1(i), PARTIC: x2(i)),\\ &\qquad\quad GO: z(i)),\\ &IN: G(x1(m..n), x2(m..n)),\\ &OUT: \neg\, m \leqq i \leqq n \vee\\ &\quad G(OBJ: proc(OBJ: x1(i), PARTIC: x2(i)),\\ &\quad LOC: z), \end{aligned} \quad (7)$$

ただし *PROC* は，i が m から n まで $proc(i)$ を繰り返すことを表し *OUT* は，$m \leqq i \leqq n$ なるすべての i について $G(proc(OBJ: x1(i), PARTIC: x2(i)))$ の関係が成り立つことを表す．なお，繰返し回数が陽に定まっていないときには，処理対象 $x(i)$ が $t(x(i))$

なるテスト条件をみたす間繰り返すものとし，手続き的表現は *WHILE* (*COND*: $t(x(i))$, *OBJ*: *proc*($x(i)$))，出力表現は$\neg t(x(i)) \vee G(proc(x(i)))$などと表す．

〔2.2〕 直列繰返し型

これは，繰返しごとに前の繰返しサイクルの結果を他のいくつかのデータとともに入力データとして用いる型であり，繰返し回数が陽に定まっている(a)の場合とそうでない(b)の場合とに大別される．

(a) 直列 *FOR* 型

PROC: *FOR*(*INDEX*: i,
FROM: m, *TO*: n,
OBJ: *proc*(*OBJ*: $x(i)$,
PARTIC: y, *GO*: y),
GO: z),
IN: $G(x(m..n), y)$,
OUT: $y(m-1)=y$,

$$\neg\, m \leqq i \leqq n \vee G(proc(OBJ: x(i), PARTIC: y(i-1), GO: y(i)), \quad (8)$$

$z=y(n)$

(b) 直列 *WHILE* 型

繰返し回数が陽に定まっていない場合であり，*PROC* 部や *OUT* 部の条件部が $t(x(i), y)$ などとなる．

直列繰返し型の手続きは通常一つの手続き名（例えば *SUM*）をもっている．このときには(8)の見出しは

PROC: *SUM*(*OBJ*: $x(m..n)$, *GO*: z),
IN: $G(x(m..n))$,
OUT: G(*OBJ*: *SUM*(*OBJ*: $x(m..n)$), *LOC*: z), (9)

に簡単化され，(8)の *OUT* 部を具象化したものを(9)の *OP* 部に記述する．

2.4 モジュール

各モジュールは，見出しとして上述の手続的表現と入出力表現をもち，本体部として *TYPE* 部と *OP* 部をもっている．*TYPE* 部はこのモジュールの見出し部に現れるデータや変数のタイプを記述する．モジュールを用いてそれらのリンクや仕様のチェックを行うときには，タイプ部を参照して，入力条件としてモジュールが仕様に適用可能かどうかをチェックする．*OP* 部は，見出し部に対する実際の処理を示す部分で，このモジュールよりも下位の手続的表現や入出力表現を手続的な制御表現でまとめている．したがってモジュールは一つの階層系を構成する．**表1**にモジュールの二，三の例を示す．

表 1 グラフ出力モジュールの一部
Table 1 Parts of graph printing modules.

```
PROC:PRINTGRAPH(
        OBJ:ORDERED_SET(f(x(m)..x(n))),
        PARTIC:x(m..n),FORMAT:form,
        GRAPH_RANGE:X_RANGE(gx_min,gx_max);
                    Y_RANGE(gy_min,gy_max)),
IN:GIVEN(OBJ:x(m..n));
   GIVEN(OBJ:f(x(m)..x(n)),LOC:y(m..n));
   GIVEN(OBJ:MAX(OBJ:y(m..n),LOC:maxy));
.....................................,
OUT:GIVEN(
       OBJ:PRINTGRAPH(
             OBJ:ORDERED_SET(f(x(m)..x(n))),
             PARTIC:x(m..n),FORMAT:form,.....)),
ENTITYTYPE:
 'gx_min,gx_max,gy_min,gy_max'(
     REG:INTEGER,STRUCT:SCALAR,VROLE:INPUT);
 x(REG:REAL,STRUCT:ARRAY(m..n),VROLE:INPUT);
.....................................,
OP:DRAW_XY_AXIS(FORMAT:STRIPE,VALUE_RANGE:....)
   FOR(INDEX:i,FROM:1,TO:n,
       OBJ:PRINTGRAPHPT(OBJ:y(i);x(i),.....)),

PROC:COMPUTE_FUNCTION(
       OBJ:ORDERED_SET(f(x(m)..x(n))),
.....................................,

PROC:PRINTGRAPHPT(
       OBJ:y(i);x(i),FORMAT:STRIPE,
       VALUE_RANGE:X_RANGE(minx,maxx);
                   Y_RANGE(miny,maxy));
       GRAPH_RANGE:X_RANGE(gx_min,gx_max);
                   Y_RANGE(gy_min,gy_max)),
```

OP 部は，見出し部に対する実際の処理を示す部分で，このモジュールよりも下位の手続的表現や入出力表現を手続的な制御表現でまとめている．したがってモジュールは一つの階層系を構成する．**表1**にモジュールの二，三の例を示す．

モジュールや仕様に現れる一般制御表現は詳細化の後，*MAPS* によりユーザ指定のプログラム言語に変換する[4]．

3. モジュール間のリンク

ユーザはライブラリモジュールの一覧表の見出しを参照[4]して仕様を作成する．仕様に書かれた手続き表現や出力表現はモジュールのインスタンスになっていることが望ましく，なければ欠けているモジュールに相当するブロックを，適当なレベルでユーザが新しく作ることが必要である．*MAPS* は仕様を始めのブロックから走査し，手続き表現は直ちに詳細化を試み，入出力表現による仕様はリンクを試み成功した後，詳細化を行う．まず，リンクの基本である縦続接続について述べ，次にブロックの先頭が制御表現の頭部である場合，フレーム内でリンクを行って詳細化を進める

方法について述べる.

3.1 クローズ形

式(1), (2)のようなモジュールの入出力述語表現は,

$$G(OBJ: q(\boldsymbol{x}), LOC: z) \vee \neg P(\boldsymbol{x}) \vee \neg G(\boldsymbol{x}) \quad (10)$$

のクローズ形で表すことができる. このような入出力間の関係を導くモジュールの OP 部を,

$$OP: z \cdot q := q_0(\boldsymbol{x}) \quad (11)$$

とする.

(10)の $q(\boldsymbol{x})$ は, $\forall \boldsymbol{x} \exists z Q(\boldsymbol{x}, z)$ による z のスコーレム関数であるが, これはまた(11)における実際上の処理手順を示す $q_0(\boldsymbol{x})$ の見出しである.

いま, 仕様として入出力表現, あるいは(10)のようなクローズ形

$$G(Q(A)) \vee \neg G(Q_1(A)) \vee \ldots \vee \neg G(Q_m(A)) \quad (12)$$

が与えられたものとする. このとき, 上の仕様が成立しないものとして上式を否定し,

$$\neg G(Q(A)), G(Q_1(A)), \ldots, G(Q_m(A)) \quad (13)$$

とする. このとき(13)とライブラリモジュールのクローズ形から反駁 (*refutation*) を行うことができれば, 反駁に用いたライブラリモジュールの OP 部から, 以下に述べるように仕様の OUT 部を導く手順, すなわちプログラムを合成することができる.

3.2 縦続接続

縦続接続とは, いくつかのライブラリモジュールの OP 部を直並列につないで仕様をみたす手順や中間プログラムを構成するもので, モジュールのリンクの基本となるものである.

例えば**図1**のように関数 $Q_0(A)=A$ を入力データとして与え, 関数値 $Q_7(A)$ を出力することを仕様が要求するものとする.

各ノードは, これに先行するノードの中間結果や入力データから, 有向アークに沿って処理して得られる関数や処理の値を表す. 図から $Q_0(A)$ から処理を縦続的に行って $Q_7(A)$ が得られることがわかる.

いま, 図1において, $Q_i(A)$ からの1入力をもつノード $Q_k(A)$, $Q_i(A)$ と $Q_j(A)$ からの2入力をもつノード $Q_k(A)$ に対応して, それぞれ,

$$G(q_k(x)) \vee \neg G(q_i(x)) \quad (14)$$

$$G(q_k(x)) \vee \neg G(q_i(x)) \vee \neg G(q_j(x)) \quad (15)$$

なるホーンクローズ形の入出力関係をもつライブラリモジュールが存在するものとする.

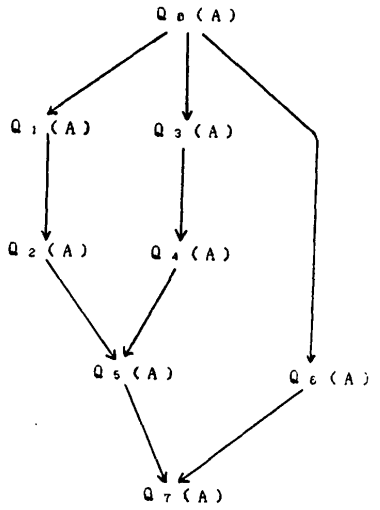


図1 縦続接続のデータフロー
Fig. 1 A data-flow of a sequential link.

このとき上記のような仕様が与えられると, その結論の否定 $\neg G(Q_7(A))$ をトップクローズとする入力反駁 (*input refutation*) を求め, 反駁に用いたライブラリモジュールを用いて, 中間プログラムを生成することができる.

いくつかの入力データから, ある出力データを与えるモジュールが存在するとき, その入力データの組を親, 出力データの組を子とよぶこととする. このとき図1に現れるデータの親子関係を用いて, 次のように中間プログラムを合成する.

(1) 入出力関係をクローズ形に変換する.
(2) 仕様の出力述語の否定を入力述語とモジュールのクローズにより反駁する. また, 仕様とモジュールのタイプ部を比較しチェックする.
(3) 反駁に用いた単一化代入によりモジュールの OP 部を具象化 (*instantiate*) する.
(4) OP 部の具象例 (*instance*) の列をデータの親子関係により並べる.
(5) 関数記号 f をもつ関数項を, 引用による能率化と人間の理解のしやすさのために, 中間変数 $Yi \cdot f$ で置き換える. ただし, i ($i=1,2,\cdots$) は変数の衝突を防止するために必要に応じてシステムが生成した番号である.

リンクに必要なモジュールがライブラリに欠けているときには, 反駁試行のトレースから不足モジュールなどを同定する.

〔例1〕 実数の列(ここでは簡単にベクトル $\boldsymbol{x}$ で表す)の個数, 和, 平均, 分散などを与える各ライブラ

リモジュールのクローズ形の入出力表現および *OP* 部を次の①～④とする．

$G(NUM(\boldsymbol{x})) \vee \neg G(\boldsymbol{x})$,
$OP: y \cdot NUM := NUM(\boldsymbol{x})$; ①

$G(SUM(\boldsymbol{x})) \vee \neg G(\boldsymbol{x})$,
$OP: y \cdot SUM := SUM(\boldsymbol{x})$; ②

$G(AV(\boldsymbol{x})) \vee \neg G(SUM(\boldsymbol{x})) \vee \neg G(NUM(\boldsymbol{x}))$
$OP: y \cdot AV := AV(SUM(\boldsymbol{x}), NUM(\boldsymbol{x}))$; ③

$G(DISP(\boldsymbol{x})) \vee \neg G(AV(\boldsymbol{x})) \vee \neg G(NUM(\boldsymbol{x})) \vee \neg G(\boldsymbol{x})$
$OP: y \cdot DISP := DISP(AV(\boldsymbol{x}), NUM(\boldsymbol{x}), \boldsymbol{x})$; ④

さて，仕様として，データ *A* を与えて分散を求める問題 (*IN: G(A), OUT: G(DISP(A))*) が与えられたものとする．まず問題のクローズ形 $G(DISP(A)) \vee \neg G(A)$ の否定から

$\neg G(DISP(A))$……⑤, $G(A)$……⑥

を得る．次に⑤をトップクローズとする反駁を行う．

図 2 仕様の否定の反駁
Fig. 2 A refutation of the negated specification.

そして反駁に用いたモジュール①，②，③，④から，データの親子関係の組を作り，これよりモジュールの *OP* 部を出力順に並べ，さらに，引数に現れる *SUM(A), NUM(A), AV(A), DISP(A)* を *Y・SUM, Y・NUM, Y・AV, Z・DISP* により置き換えることにより，次のようなプログラムの中間表現を得る．

$Y \cdot NUM := NUM(A)$;
$Y \cdot SUM := SUM(A)$;
$Y \cdot AV := AV(Y \cdot SUM, Y \cdot NUM)$;
$Z \cdot DISP := DISP(Y \cdot AV, Y \cdot NUM, A)$; ⑦

3.3 条件分岐フレーム内のリンク

(5)の形の条件分岐を含む仕様に対しては，詳細化の問題は，次のような分岐表現のフレーム内の *OBJ* 部におけるリンクの問題に簡単化できる．

IF_THEN(COND:t $(\boldsymbol{x})$,
OBJ: (*IN:* $G(\boldsymbol{x})$, *OUT:* $G(q(\boldsymbol{x}))$),
GO: z) (16)

(6)の *CASE* 文の場合も同様で，各々の *OBJ* 格を上式の *OBJ* 格の形に整備すればよい．

始めから(16)の形で仕様が与えられず(5)や(6)のような形で仕様が与えられたときには，この形の見出しをもち(16)の *OP* 部をもつモジュールを設けておき，詳細化のときこれで置き換える．

3.4 繰返しフレーム内のリンク

仕様に繰返しを含む場合，繰返しのブロックを一つのモジュールとみて他のモジュールとリンクするのは，例1のように問題はない．しかし，繰返しを制御する構成要素のレベル，例えばインデクス変数の開始値や増減の調整といったレベルから，一般的かつ能率的にリンクを行うことは，困難である．ここでは，条件分岐の場合と同様に，繰返しのブロックの詳細化の問題を繰返しのフレームの中の *OBJ* 格の問題に簡単化して詳細化する．

条件分岐の場合と同様に，(7)や(8)の見出しをもつ並列型，直列型の繰返しモジュールに対し，次のような *OP* 部を設けておき，(7)や(8)の形の仕様が与えられたときこれで置き換えて詳細化する．

(1) 並 列 型

OP: FOR(INDEX: i, FROM: m,

```
        TO: n, OBJ:
    (IN: G(x1(i), x2(i)),
     OUT: G(proc(OBJ: x1(i),
                 PARTIC: x2(i))),
          GO: z(i))                    (17)
```

(2) 直 列 型

直列 *FOR* 型

```
OP: FOR(INDEX: i, FROM: m, TO: n,
    OBJ: (IN: G(x(i), y(i−1)),
          OUT: G(proc(OBJ: x(i),
                      PARTIC: y(i−1),
                      GO: y(i)))), GO: z)
                                       (18)
```

WHILE 型は，*OP* 部の繰返し終了の判定条件が $t(x(i), y)$ などに代わる以外は *FOR* 型と同じである．

3.5 繰返しの統合

モジュールは通常，原則として一出力だけをもつように作るので，モジュールを組み合わせて仕様を作るときには，同じ繰返し回数のループが縦続する形になることが多い．そこで仕様の詳細化途中においてこのようなループの縦続を統合してプログラムの効率を改善する．

いま，仕様からモジュールの *OP* 部を展開して次のような二つの *FOR* 文の表現を得たものとする．

```
Z1 := Z10;
FOR(INDEX: I, FROM: M, TO: N,
    OBJ: Z1 := Q1(X(I), Z1));
Z2 := Z20;
FOR(INDEX: J, FROM: M, TO: N,
    OBJ: Z2 := Q2(X(J), Z2));
```

上式においてどの *FOR* 文も繰返し制御変数の値の区間が同じであり，また互いに他の出力変数の値を入力データとして必要としない．このようなときには，これらを統合して

```
Z1 := Z10; Z2 := Z20;
FOR(INDEX: I, FROM: M, TO: N,
    OBJ: (Z1 := Q1(X(I), Z1);
          Z2 := Q2(X(I), Z2)));
```

とすることにより，実行時間と所要記憶容量を減少させることができる．

繰返しが *WHILE* 文によって構成されている場合にも，条件が等しく，互いに他の出力変数の値を入力データとして要求しない場合には二つの *WHILE* 文は，*FOR* の場合と同様に一つに統合することができる．付図1にこの方法を例1に適用してC言語に展開した結果を示す．

4. 仕様のチェック

この章では3章の始めに述べたような方法でユーザが作成した仕様に生ずるいろいろな誤りを，ライブラリモジュールなどを用いて検出する方法について述べる．

仕様が次のような手続き表現やタイプ表現の列で与えられているとする．

```
PROC1 (OBJ: X1,...,GO: Z1);
    .....                              (19)
PROCN(OBJ: XN,...,GO: ZN);
```

システムはまず仕様の手続き名などを用いて各手続き表現に適用可能なライブラリモジュールを探索し確認する．次にこれを(19)のような仕様の手続き表現と単一化し，単一化代入により具象化した各モジュールの入出力表現やタイプ部で仕様の手続き表現を置き換える．

単一化不可能な場合は，書式の部分的誤りや手続き表現の必須格の値の指定の脱落などの場合であり，ユーザに引数などを問合わせることにより，訂正したり不足部分を補って単一化を行う．

このようにして作った入出力表現の列を，クローズ表現に変換してつぎのような入出力クローズの列をつくる．

```
G(OBJ: F1(X1), LOC: z・F1) ∨ ¬ G(X1)
                ∨ ¬ P1(X1);
R1(F1(X1)) ∨ ¬ G(X1) ∨ ¬ P1(X1);
    .....                              (20)
G(OBJ: FN(XN), LOC: z・FN) ∨ ¬
                G(XN) ∨ ¬ PN(XN);
RN(FN(XN)) ∨ ¬ G(XN) ∨ ¬ PN(XN);
```

ただし，$P1(X1), \cdots, PN(XN)$ は入力の性質やモジュールのタイプ部から作成される型を表す述語などであり，$R1(Z1), \cdots, RN(ZN)$ も出力についての同様の述語とする．また，$F1, \cdots, FN$ は $PROC1, \cdots, PROCN$ に対応する関数で，引数に IN, OUT を含むときには同様の方法で引数部をチェックする．

ユーザが与えた入力データとその段階までに得られている中間結果の出力データを用いて，(20)の各入出力表現が入出力条件，入力変数の性質やタイプなどを満たしているか否かを，各入出力表現ごとに反駁法を用いてチェックする．以下にその具体的手順について

述べる.

〔手順1〕

各入出力クローズ Ci について，クローズ番号 i の順に次の処理を行う.

クローズの集合の格納場所を S とし，始めに S を空にしておく.

(1) ファイルやメモリ上にユーザの与えたデータがあれば，これをその性質やタイプとともに S に加える.

(2) i 番目の入出力表現の出力述語を否定し，これを(20)の i 番目の入出力クローズ形と S により反駁する. ただし，プリミティブな入力データは与えられたものとして S に加える.

(3) 反駁に成功したら出力述語を S に加える.

(4) 反駁に成功しない場合には入出力クローズの中で空にならずに残っているものをユーザに提示する. なお修正せずにチェックの処理を続行する場合には反駁に必要とされるクローズを仮定して出力述語を S に加える.

上記の手順により反駁が不成功になるのは次の場合である.

(A) ¬ $G(X)$ の形のクローズが残る場合,

(1) 入力データが与えられていない.

(2) ¬ $G(X)$ の X が $F(X1)$ のような関数の場合，$X1$ から $F(X1)$ を得るための前処理が欠けている.

(3) 繰返しの初期値が設定されていない.

などであり,

(B) ¬ $P(x)$ の形のクローズが残っている場合には,

(1) 入力データがモジュールの入力データの型と一致しない.

(2) 前処理などが欠けており入力データが所要の性質を満たしていない.

などである.

〔例2〕「$X(I)$ のデータ ($I=0,1,\cdots,20$) を $X(0..20)$ に読み込み，$ORDERED\_SET(F(X(0)..X(20)))$ を $X(0..20)$ に対し $X\_RANGE(100,400)$, $Y\_RANGE(100,400)$ の範囲の画面上に棒グラフの形式で表示せよ.」に対応する形式仕様は次のようになる.

READARRAY1(OBJ: INPUT_FLLE,
GO: X(0..20)); ①
PRINT_GRAPH
(OBJ: ORDERED_SET(F(X(0)..X(20))),
PARTIC: X(0..20), FORMAT: STRIPE,
GRAPH_RANGE: X_RANGE(100, 400);
Y_RANGE(100, 400)); ②

これらの手続き表現をモジュールと単一化して入出力表現に変換することにより,

G(OBJ: X(0..20)) ∨
¬ G(OBJ: INPUT_FILE); ③
G(OBJ: PRINT_GRAPH)
OBJ: ORDERED_SET(F(X(0)..X(20))),
PARTIC: X(0..20), FORMAT:
STRIPE,
GRAPH_RANGE: X_RANGE(100,400);
Y_RANGE(100,400)))
∨ ¬ G(OBJ: X(0..20))
∨ ¬ G(OBJ: ORDERED_SET
(F(X(0)..X(20))))
∨ ¬ G(OBJ: MAX(ORDERED_SET
(F(X(0)..X(20))))
∨ ¬ G(OBJ: MIN(ORDERED_SET
(F(X(0)..X(20))))
∨ ¬ G(OBJ: MAX(X(0..20)))
∨ ¬ G(OBJ: MIN(X(0..20))) ④

を得る. 上記の入出力表現の列に手順1を適用することにより，①，②の入出力表現における出力述語の否定の反駁に失敗してクローズ

¬ G(OBJ: INPUT_FILE),
¬ G(MAX(OBJ: ORDERED_SET
(F(X(0)..X(20)))))
¬ G(MIN(OBJ: ORDERED_SET
(F(X(0)..X(20)))))
¬ G(MAX(OBJ: X(0..20)))
¬ G(MIN(OBJ: X(0..20)))

が残る. システムはこれらをユーザに提示して前処理などの付加を要求する.

(A)(3)で述べた繰返しの初期値の未設定は，直列繰返しの場合に生じるものである. 直列繰返しは，次のようなプログラムの制御パタンをもつ.

Y := Y0;
FOR(INDEX: I, FROM: M, TO: N,
OBJ: ⋯; Y := F1(Y, X(I)); ⋯) (21)

これに対応する入出力クローズは,

G(OBJ: Y);

$G(IN: G(Y, X(I)),$
$OUT: \cdots; G(OBJ: F1(Y, X(I)),$
$LOC: Y); \cdots)$
$\vee \neg M \leqq I \leqq N$
$\vee \neg G(OBJ: X(M..N))$
$\vee \neg G(OBJ: Y);$ (22)

であり，これと与えられたデータ条件を用いて仕様の出力述語の否定のクローズを反駁することができる．このとき，初期値設定が欠けていれば，

$\neg G(OBJ: Y)$ (23)

の形のクローズが残って反駁に失敗する．

初期値設定の欠落は，手続き表現(19)のままでも可能と考えられるが，初期値設定と *FOR* 文が離れている場合には，単なるパタンマッチングによっては検出できない．一方，手順1に従って上記のようにクローズ形にして反駁法を用いれば，初期値設定が *FOR* 文と離れていても問題はない．

なお，手順1のような誤り検出の手続きに，別の手続きを加えて仕様の中の，異なる誤りパタンを検出することができる．例えば多重繰返し制御におけるインデクス変数の衝突の検出である．そのためには，繰返し制御表現の見出し部が現れるごとに，繰返しのインデクス変数とループのレベルの深さをツリー状にレコードし，ループが終るごとにレコードのその部分を消すものとする．このとき，各繰返し表現の見出し部においてインデクス変数が，根ノード(レベル1のノード)の道の上のインデクス変数と衝突しないかどうかをチェックすればよい．

5. システム構成と実験

以上に述べた方式に基づいてモジュールのリンクと仕様のチェックを対象として，プログラムの詳細化展開システム *MAPS* を作成した．システムの主要な部分は，*Prolog* に分岐，繰返しの処理用のプログラムを付加したリンクシステムと，リンクシステムの出力結果をユーザが指定したオブジェクト言語に変換する詳細化展開システム(*LISP* で約 9k セル)と仕様のチェックシステム，基本モジュール(約30個)および応用モジュール(表操作，文字列処理など，約40個)からなる．**図3**にシステムの動作図を示す．図3において実線は処理の流れを示し破線はデータの流れを示す．

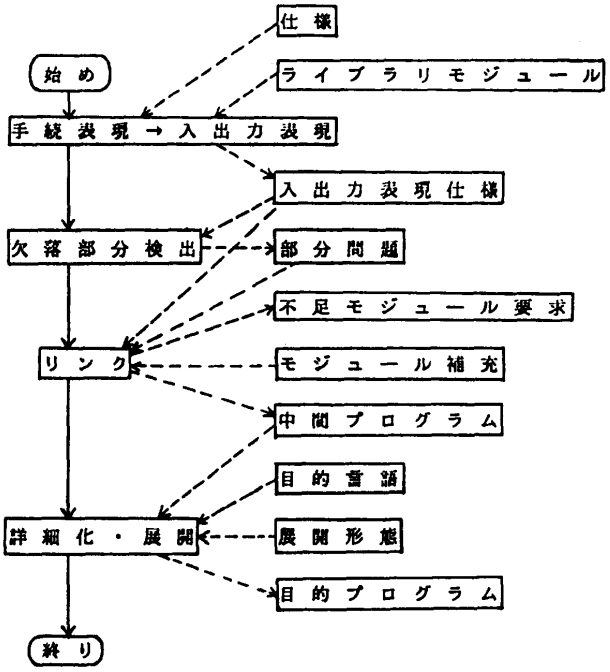


図3 システムの動作
Fig. 3 The flow of the system.

```
IN: GIVEN(OBJ:X(0..20));
OUT:GIVEN(OBJ:PRINTGRAPH(
                OBJ:ORDERED_SET(F(X(0)..X(20))),
                PARTIC:X(0..20),FORMAT:STRIPE,
                GRAPH_RANGE:X_RANGE(100,400);
                            Y_RANGE(100,400)))
```

図4 形式仕様
Fig. 4 A formal specification.

```
COMPUTE_FUNCTION(OBJ:ORDERED_SET(F(X(0)..X(20))),
                 GO:Y(0..20));
MAX(OBJ:Y(0..20),GO:MAXY);
MIN(OBJ:Y(0..20),GO:MINY);
:=(MAXX,X(20));
:=(MINX,X(0));
PRINTGRAPH(OBJ:ORDERED_SET(F(X(0)..X(20))),
           PARTIC:X(0..20),FORMAT:STRIPE,
           GRAPH_RANGE:X_RANGE(100,400);
                       Y_RANGE(100,400))
```

図5 グラフ出力の中間プログラム
Fig. 5 An intermediate form of the graph printing program.

以下に，2章表1で示した *PRINT_GRAPH, MAX* などのモジュールにより関数のグラフを描くプログラムを生成する例を示す．プログラムの仕様を次に示す．

「*IN: X*(0..20) が与えられる．
OUT: ORDERED_SET(*F*(*X*(0)..*X*(20))) を

X(0..20) に対し *X_RANGE* (100, 400), *Y_RANGE* (100, 400) の範囲の画面上に棒グラフの形式で表示したものが与えられる.」

この仕様の形式表現を図4に示す. 図4の形式仕様を表1のモジュールや, 関数計算のモジュール, 最大値計算のモジュールなどを用いてリンクすることにより図5の中間プログラムを求め, これを詳細化展開システムに補助情報 (目的プログラム言語, 副プログラムとして呼び出すか主プログラムの中に埋め込むかの指示) とともに入力して図6(a)の *LISP* プログラムを得る. また, このプログラムの動作結果の例を同図6(b)に示す.

処理時間は ACOS-850 のインタプリタモードの *LISP* と *Prolog* で, リンクに数秒, 詳細化と展開に数十秒の程度である.

上記のほかに, 男女別平均, 文字列置換えプログラム, 最小二乗法による直線の当てはめプログラム, 分数計算プログラム, データのマージソートプログラムなどをモジュールのリンクにより生成したがいずれも上例と同程度の処理時間であった.

6. むすび

前報告に続き本報告では, ライブラリモジュールを用いて, リンクの手法により仕様の誤りをチェックしたり, 仕様を詳細化する方法を述べた. 後者は人手で詳細化やコーディングを行う場合と比べ, 次のような利点がある.

(1) ライブラリモジュールの表を参照することによりプログラムの設計の指針が得られる.
(2) 詳細なアルゴリズムを記述しなくてよい.
(3) 詳細な手続きを目的言語で記述する必要がないためコーディングの誤りが発生しない.

今後, より汎用的でフレキシブルなライブラリモジュールの作成法, 詳細化過程における大域的最適化や自然言語的表現の仕様への導入などの問題がある.

謝辞 本研究の, システムの作成と実験に協力された塩野入理君 (現在 NTT) に深謝いたします.

```
(defun f (x)
 (plus (times 0.32 (difference x 10) (difference x 15) x) 26))

(array y t 21) (array x t 21)
(defun printgraph ()
 (prog (i20l x maxx maxy i4l i1l i11l miny minx xy_pos)
  (setq i1l 1)
  (loop nil (store (y i1l) (f (x i1l)))
            (setq i1l (add1 i1l))
            (cond ((greaterp i1l 20) (exit-loop))))
  (setq maxy (y 1)) (setq i4l (plus 1 1))
  (loop nil (cond ((greaterp (y i4l) maxy) (setq maxy (y i4l))))
            (setq i4l (add1 i4l))
            (cond ((greaterp i4l 20) (exit-loop))))
  ..................................................
  (cond ((and (lessp miny 0) (lessp 0 maxy))
         (drawline 100 gy_pos21l 400 gy_pos21l)))
  (cond ((and (lessp minx 0) (lessp 0 maxx))
         (drawline gx_pos21l 100 gx_pos21l 400)))
  (draw_box 100 400 100 400)
  (setq i20l 1)
  (loop nil
   (setq xy_pos (find_xy_gposition (x i20l) (y i20l)
                  maxx minx maxy miny 100 400 100 400))
   (setq x_pos39l (car xy_pos))
   (setq y_pos39l (cadr xy_pos))
   (plot x_pos39l y_pos39l 100 'stripe)
   (setq i20l (add1 i20l))
   (cond ((greaterp i20l 20) (exit-loop))))))
```

図 6(a) 展開結果
Fig. 6(a) The result of expansion.

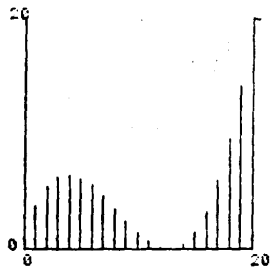


図 6(b) 実行結果
Fig. 6(b) A result of execution.

参考文献

1) Teichrow, D. and Hershey, F. A.: PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing System, *IEEE Trans. Softw. Eng.*, Vol. SE-3, pp. 41-48 (1977).
2) Darlington, J. L.: Automatic Synthesis of SNOBOL Programs, in Simon, J. C. (ed.), *Computer Oriented Learning Process*, pp. 443-453, Nordhoffleyden (1976).
3) 藤田, 西田: 階層的定義系によるプログラム生成, *IECE Jpn. Trans.*, Vol. J 61-D, No. 2, pp. 103-110 (1978).
4) 西田, 藤田: ライブラリモジュールを用いたプログラムの半自動的詳細化, 情報処理学会論文誌, Vol. 25, No. 9, pp. 785-793 (1984).
5) 西田: 言語情報処理, 第2刷, コロナ社, 東京 (1984).
6) 藤田, 西田, 塩野入: ライブラリモジュールのリンクによるプログラム生成, 第 29 回情報処理学会全国大会論文集, 2R-3 (1984).

7) Nishida, F., Fujita, Y. and Takamatsu, S.: Construction of a Modular and Portable Translation System, *Proceedings of COLING '86*, pp. 649-651 (Aug. 1986).
8) 西田，藤田，高松：日本語による仕様記述からのライブラリモジュール援用プログラムの半自動生成，情報処理学会（プロトタイピングと要求定義）シンポジゥム論文集，pp. 111-119 (Apr. 1986).

```
double  disp (A ) double A [ 100 ] :
   {double Y.NUM :
    double Y.SUM :
    double Y.AV :
    double Z.DISP :
    double y :
    double y1 :
    int i :
    int j :
    Y.NUM = 0.0 :
    Y.SUM = 0.0 :
    for (i = 1 ; i <= 100 : ++ i)
     {
      Y.NUM = Y.NUM + 1 :
      Y.SUM = Y.SUM + A [ i ] :
     } :
    Y.AV = (Y.SUM) / (((Y.NUM - 1) + 1) ) :
    y = 0.0 :
    for (j = 1 ; j <= 100 : ++ j )
     {
      y1 = (A [ j ] - Y.AV ) :
      y = (y + y1 ) :
     } :
    Z.DISP = (y ) / (((Y.NUM - 1) + 1) ) :
   }
```

付図 1 最適化結果
Fig. A1 An optimized result.

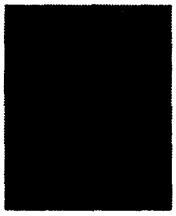
(昭和 61 年 11 月 20 日受付)
(昭和 62 年 2 月 12 日採録)

西田富士夫（正会員）

大正 15 年生．昭和 25 年京都大学工学部電気工学科卒業．現在，大阪府立大学工学部電気工学科教授．工学博士．自然言語処理，プログラムの仕様と詳細化，問題解決システムなどの研究に従事．著書「言語情報処理」など．電子情報通信学会，電気学会，計測制御学会，IEEE 各会員．

藤田 米春（正会員）

昭和 21 年生．昭和 43 年大阪大学基礎工学部電気工学科卒業．昭和 48 年同大学院博士課程修了．工学博士．同年大阪府立大学工学部電気工学科助手．62 年 4 月より，大分大学工学部助教授．記号処理，プログラム生成，問題解決などの研究に従事．電子情報通信学会会員．



高松 忍（正会員）

昭和 23 年生．昭和 46 年大阪府立大学工学部電気工学科卒業．昭和 48 年同大学院工学研究科修士課程修了．工学博士．現在，大阪府立大学工学部電気工学科助手．自然言語処理，知識情報処理の研究に従事．電子情報通信学会，人工知能学会，日本認知科学会各会員．