Vol. 25 No. 5　　情報処理学会論文誌　　Sep. 1984

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†

西田 富士夫††　藤田 米春††

本論文は，あらかじめ構成されている階層的なライブラリモジュール[5]を用いて，仕様からプログラムを半自動的に生成する手法について述べている．仕様は，手順的表現，入出力述語による表現およびプログラミング言語表現の3種類の表現を併用して書かれ，手順的表現と入出力述語による表現には，各引数に格名をつけてそれぞれの役割を明確にしている．本手法に基づくプログラム生成システムSAPREは，まず与えられた仕様の各変数の因果関係をチェックし，次にユーザと対話しながらライブラリモジュールを用いて仕様を詳細化し，最終的にユーザ指定の言語のプログラムに変換する．SAPREは，ライブラリモジュールの適用において，簡単化した2階の単一化手順を用いている．

1. まえがき

近年，ソフトウェアの生産性，信頼性を上げる方法として，要求仕様技術[3]やプログラムの自動合成[2]が研究されているが，実用上，解決すべき問題がまだ多いようである．一方，多数の類似のプログラムが作成されているが，これらは十分に再利用されていないようである．しかし，類似のプログラムを統合し，一般化してライブラリとして蓄え，これを用いて仕様に合うようにトップダウン的にモジュールをリンクし詳細化し変形していくことにすれば，能率的かつ柔軟にプログラムに展開できると考えられる[4),6),7)]．

このような考え方に従って，この報告では，プログラムの仕様からライブラリモジュールを用いて半自動的に仕様を詳細化し，プログラムに変換する方法を述べる．仕様は，格ラベルをもつ手順的表現か入出力関係による述語的表現を用いて表し，人間にも機械にも取り扱いやすいものとしている．

現在拡張中の実験システムSAPRE(Semi-Automatic Program Refinement and Expansion system)は，ユーザとの対話により仕様を詳細化し，これをライブラリモジュールを用いて具体的なプログラムに変換する．各ライブラリモジュールには，その手順をやや詳細に示すOP部があり，OP部は詳細化中にユーザに情報を要求する文を含んでいる．また，これらのモジュールを仕様などに適用するために，2階の論理における単一化[1),8)]を簡単化したものを用いている．

2. プログラムの仕様

プログラムの仕様記述はいろいろな表現を用いて行うことができるが，仕様の内容により，それらの表現の適不適が変わってくる．たとえば，方程式などのように，入力と出力の関係が陽に与えられるものは，入出力関係による仕様記述が適当である．他方，ファイルの更新，合併やオンラインで応答を行う事務用のシステムなどは，手順的表現のほうが表現が容易であり，適切な場合が多い．このような観点から，この報告では，入出力表現，手順的表現および，演算制御をプログラムレベルで指示するプログラム表現の3種類を仕様記述に適宜用いることにする．

2.1 手順的表現

この表現は，

処理名（格名：ターム，…，格名：ターム）　(1)

の形をしている．ここにタームとは変数，配列，集合，関数，論理式などからなり，処理名は，これらのタームを処理する手順名である．手順的表現は，PASCALなどの procedure や function と類似の形をしているが，(1)式は，PASCALなどでは引数として現われない条件なども含む．また，格名を前置しているので，手順的表現の処理内容がより明確になる．ただし，(1)式はタームに配列や論理式などを含むので，後述する単一化手順は2階の単一化手順を必要とする．

次に格ラベルとして標準的なものについて簡単に説明する．

OBJect：手順が適用される対象となるタームを

† Semi-Automatic Program Refinement from Specification Using Library Modules by FUJIO NISHIDA and YONEHARU FUJITA (Department of Electrical Engineering, College of Engineering, University of Osaka Prefecture).

†† 大阪府立大学工学部電気工学教室

入れる格

SOurce : OBJ 格のタームが属する集合や配列を入れる格

CONDition : OBJ 格のタームの処理条件を入れる格

GOAL : 手順の適用結果を格納する変数を入れる格

KEY : OBJ 格のタームを処理するために用いるキーパラメータを入れる格

PARTICipant : OBJ 格のタームの処理に関係するタームを入れる格

上記の他に処理手順の動作モードを指定するためのMODE 格やプログラム言語表現で用いる COUNT, FROM, TO などの格を設ける．また，変数の型を記述する ENTITYTYPE 部では，REGion (変数の値域)，STRUCTure (変数の構造)，Variable ROLE (入力変数，出力変数の区別) などの格を設ける．これらの格名は，タームの役割が明白な場合には読みやすくするため省略することがある．

［例1］ 配列 $t(1..n, 1..m)$ から第1列目が r である行をすべて見つけ出して，配列 $ans(1..n, 1..m)$ に入れる処理の手順的表現は，

SEARCH (SO : $t(1..i..n, 1..m)$,
OBJ : $t(i, 1..m)$, COND : $t(i, 1)=r$,
GOAL : $ans(1..n, 1..m)$)

で表される．

この手順的表現は，人間と機械の両方に理解しやすい一つのニーモニックな仕様記述として導入される．

2.2 入出力表現

入出力表現は次のような条件の対である．

(IN : $P(x)$, OUT : $Q(x, z)$) (2)

ここに $P(x)$, $Q(x, z)$ は，通常，連言標準形か

$$\bigwedge_i(\bigwedge_i P_{ij}\rightarrow Q_i)$$

の形をとるものとし，原則として手順的表現と同様に格名を引数に前置するものとする．また，格名 IN および OUT は，後述するモジュールにおける PROC, ENTITYTYPE, OP とともに，モジュールや仕様記述の枠組内のラベルとして用いる．

［例2］ 5000 人の8科目のテストの得点の表 $SCORE(1..5000, 1..8)$ において，各人の8科目の点の合計を SCORE の9列目に格納するプログラムの仕様．

(IN : GIVEN (OBJ : $SCORE(1..5000, 1..8)$),
OUT : $\forall i\in(1..5000)(SCORE(i, 9)$
$=SUM(SCORE(i, 1..8))))$.

ここに，GIVEN (OBJ : t) の形の表現は，t が入力データとして，あるいは中間計算結果として与えられていることを表す．

2.3 プログラム表現

仕様記述において，仕様の各部分的処理がたんに処理の順に一列に連なっている場合や，隣り合う各部分的処理の間にすきまが存在しない場合には，手順的表現や入出力表現のみで仕様を記述することができ，問題は生じない．しかし，仕様記述のなかで分岐や繰返しを指示する必要がある場合や，部分的処理を表すモジュールの入出力表現や手順的表現の間に細かい処理を挿入する場合には，プログラミング言語の制御ステートメントによる直接的な表現などが有用な場合が多い．このような場合には，目的言語の制御ステートメントなどで直接に表現するか，または後述の一般演算制御言語（一般言語）のステートメントを用いてひとまず，仕様を表現しておき，これを変換モジュールを用いて目的言語に変換する．

2.4 仕様のモジュールへの展開

ライブラリモジュールのなかに，与えられた仕様を包含するものが存在すれば，仕様の作成はこのモジュールに具体的なパラメータ値を与えることにより完了する．しかし，実際の仕様に対して，そのようなモジュールが存在することは少なく，いくつかのモジュールを組み合わせて仕様をみたし，詳細化を行うことが多い．以下このような手順についてその概要を述べる．

まず，ユーザはシステムに，作成しようとしているプログラムの関連分野の名前（統計処理，記号処理，テーブル操作，予約・在庫処理など）を入力し，関連分野のモジュールの提示を求める．次に，システムが提示したモジュール群から与えられた仕様のなかの主要な処理内容を含むモジュールを選び出して，モジュールの手順的表現を処理の順にならべる．このとき手順的表現の SOurce 格や OBJect 格などのタームは，前段までのモジュールの GOAL 格などで与えられ，因果律をみたしていなければならない．モジュールのパラメータで因果関係に関与しないパラメータが未定義の場合には，これを省略しておき，後に詳細化の段階でシステムの要求によりこれらのパラメータを入力する．このように並べたモジュール間にすきまがある場合には，入出力表現やプログラム表現を用いてすきまをカバーする．こうしてできた第一段階の仕様

をシステムに入力すると，システムは，モジュールの手順名のチェックや因果関係のチェックを行い，ユーザに不完全な部分の修正を要求する．さらに，仕様の中の入出力表現とマッチする入出力表現をもつモジュールがあれば，このモジュールの手順的表現で，入出力表現を置き換える．入出力表現の出力述語または入力述語のいずれか一方のみがマッチするモジュールが見つかった場合には，そのモジュールの手順的表現と，これと縦続する残りの部分問題を表す入出力表現に分解する．ユーザは，この結果得られる仕様のなかで，対応するモジュールが存在しない入出力表現の部分を目的言語などを用いて書き改めて，最終的な仕様記述とする．

仕様に適用されたモジュールやその OP 部に未指定のパラメータや選択項目を含むときには，システムからユーザへの質問回答により仕様を詳細化する．

表 1　プログラム変換モジュールの一部
Table 1　A part of the common language modules.

```
(1) PROC: IF-THEN (COND: *pred, OP: *spec),
    ENTITYTYPE: *pred (REG: BOOLEAN, STRUCT: PRED);
                *spec (STRUCT: SPEC),
(以下では，*spec, *spec1, *spec2 や *pred の ENTITYTYPE は上記と同様であるので省略する.)
    IN: GIVEN (OBJ: *pred, *spec),
    OUT: (*pred→STATE (*spec))
        ∧(NOT (*pred)→STATE (NO-OPERATION)),
    OP: (LISP (COND (*pred *spec)));
        (PASCAL IF *pred THEN *spec),
(2) PROC: IF-THEN-ELSE (COND1: *pred, OP1: *spec1,
                        COND2: NOT (*pred), OP2: *spec2),
    .....
    OP: (LISP (COND (*pred *spec1) (T *spec2)))
        (PASCAL IF *pred THEN *spec1 ELSE *spec2),
(3) PROC: FOR (COUNT: *I, FROM: *M, TO: *N, OP: *spec),
    .....
    OP: (LISP (SETQ *I *M)
        (LOOP ( ) (COND ((EQVAL *I *N)
                         (EXIT-LOOP)))
                  *spec (SETQ *I (ADD1 *I))));
        (PASCAL FOR *I := *M TO *N DO *spec),
```

3. ライブラリモジュールと単一化

3.1 ライブラリモジュール

ライブラリモジュールは基本的なプログラムモジュールを用いて階層的に作成される．またこれらのモジュールは，分野ごと，たとえば統計処理，テーブル処理，入出力操作などに分類されて格納される．ライブラリの各モジュールは，見出し部と型部および OP 部からなる．見出し部は，手順的表現と入出力表現とからなり，この 2 種類の表現のどちらを用いても検索可能である．型部 (ENTITYTYPE 部) は，モジュールで用いられる変数の値域 (REGion)，構造 (STRUCTure) および入力変数か出力変数かなど変数の役割 (Variable ROLE) を記述している．なお，単一化において他のタームの代入が許される変数には "*" を前置して仕様の変数と区別する．OP 部は，見出し部で表されたモジュールの機能を実現するために手順をやや詳細に表したもので，数個のより下位のモジュールあるいはプログラム表現を用いて書かれている．ただし，プログラム表現は，プログラミング言語をある範囲で自由に選べるように，**表 1** の PROC 部のように一般的なプログラム表現 (一般言語) で表しておき，展開時に OP 部をユーザ指定の目的言語のプログラムでおきかえるものとする．表 1 において，*STATE* (*spec) は，*spec に対応するプログラムが実行された後の状態を表す述語である．

モジュールの OP 部には，ユーザ指定の目的言語に合わせて選択される項目や，何種類かの展開方法のうち，その一つをユーザに選択させる表現 *?SELECT* $(t, t_1: S_1, \cdots, t_n: S_n)$ を設け，t の値として $t_1, \cdots, t_n$ の一つをユーザに選択させる．t_i が選択された場合には，この *?SELECT* 文を S_i によって置き替える．

表 2 にテーブル操作のモジュールの一部を示す．

表 2 において，*SEARCH* は，2 次元配列のなかから，COND 格の条件を満たす行を探し出して GOAL 格の領域に入れる操作である．このモジュールの OP 部には，ユーザに *KEY-SEARCH* (キー項目による探索) か *GENERAL-SEARCH* (一般的な探索) かの選択を要求する *?SELECT* 文が設けられており，展開に際してユーザの指示を求める．また，$\#=(x, y)$ は，x に対する y の代入操作を表しており，*WHILE* (COND: *P, OP: *spec) は，WHILE *P DO *spec の手順的表現である．*JOINKEY* はキーによる表の合併である．

3.2 単 一 化

ライブラリモジュールは従来のサブルーチンを一般化し統合したものである．このため，もとのプログラムでは定数であったものが変項化されている場合が多い．このモジュールを，与えられた仕様に合ったプロ

表 2 テーブル操作モジュールの一部
Table 2 A part of the table processing modules.

```
(1) PROC: SEARCH (SO: *x(*n1..*i..*n2, *m1..*m2),
                  OBJ: *x(*i, *m1..*m2),
                  COND: ∧*j *Pj(*x(*i, *j), *r(*j)),
                  GOAL: *z(*n1..*n2, *m1..*m2)),
    ENTITYTYPE:
      *x (REG: REAL, STRUCT: ARRAY(*n1..*n2, *m1..*m2),
          VROLE: INPUT);
      *z (REG: REAL, STRUCT: ARRAY(*n1..*n2, *m1..*m2),
          VROLE: OUTPUT);
      *r (REG: REAL, STRUCT: ARRAY(*m1..*m2),
          VROLE: INPUT);
      {*Pj | *j∈(*m1..*m2)}(REG: BOOLEAN, STRUCT: PRED,
          VROLE: INPUT);
      "*n1, *n2, *m1, *m2" (REG: INTEGER, STRUCT: SCALAR,
                            VROLE: INPUT);
      "*i, *j" (REG: INTEGER, STRUCT: SCALAR, VROLE: LOCAL).
    IN: GIVEN (OBJ: *x(*n1..*n2, *m1..*m2); *r(*m1..*m2)),
    OUT: ∀*i∈(*n1..*n2) ∧*j *Pj(*z(*i, *j), *r(*j))
        ∧∀*i∈(*n1..*n2)·*z(*i, *m1..*m2)∈*x(*n1..*n2,
                                            *m1..*m2),
    OP: ? SELECT (q1, KEY-SEARCH:
        ? SELECT (q2, BINARY-SEARCH:
            IF-THEN (COND: NOT (SORTED (OBJ: *x(*n1..*i..*n2,
                                *m1..*m2), KEY: *x(*i, 1)),
                     OP: SORT (OBJ: *x(*n1..*i..*n2, *m1..*m2),
                               KEY: *x(*i, 1),
                               GOAL: *x(*n1..*n2, *m1..*m2)));
            BINARY-SEARCH (
                SO: *x(*n1..*i..*n2, *m1..*m2),
                OBJ: *x(*i, *m1..*m2), COND: =(*x(*i, 1), *r(1)),
                KEY: *x(*i, 1), GOAL: *z(1, *m1..*m2)),
            SEQUENTIAL-SEARCH: #=(q1, GENERAL-SEARCH),
        GENERAL-SEARCH:
            SEQUENTIAL-SEARCH (
                SO: *x(*n1..*i..*n2, *m1..*m2),
                OBJ: *x(*i, *m1..*m2),
                COND: ∧*j *Pj(*x(*i, *m1..*m2), *r(*m1..*m2)),
                GOAL: *z(*n1..*n2, *m1..*m2)))
(2) PROC: JOINKEY(SO: *X(*N1)..*I1..*N21, *M11..*J1..*M21),
      PARTIC: *Y(*N12..*I2..*N22, *M12..*J2..*M22),
      OBJ: {*X(*I1, *M11..*J1..*M21),
            *Y(*I2, *M12..*J2..*M22)},
      COND: =(*X(*I1, *K1), *Y(*I2, *K2)),
      GOAL: *Z(*N13..*N23, *M13..*M23)),
        .....
    OP: FOR (COUNT: *I1, FROM: *N11, TO: *N21,
        OP: SEARCH (SO: *Y(*N12..*I2..*N22,
                           *M21..*J2..*M22),
                    OBJ: *I2,
                    COND: =(*X(*I1, *K1), *Y(*I2, *K2)),
                    GOAL: *ZI(*M21..*J2..*M22));
                    JOINT (OBJ: X(*I1, *M11..*J1..*M21),
                           .....);
                    ADD (OBJ: .....)).
```

グラムに変形するためのおもな処理は，各変項に仕様の中の具体的な値や記号を代入する単一化の処理である．現在用いられている単一化手続きは1階の範囲のものがあるが，ライブラリモジュールは一般に関数や述語が変項化されたものを含むので，2階の単一化を導入した．したがって，1階の単一化では不可能な次のような単一化代入を見つけることができる．

［例3］ 2変数の関数定項 $+(u, v)$ の u を一変数の関数変項 $*f(*x)$ の $*x$ に対応づけることにより，$+(*x, v)$ を $*f(*x)$ に代入する．（これは，ラムダ記法を用いれば，$*f \leftarrow \lambda *y \cdot +(*y, v)$ と表される．同様に，述語変項よりなるモジュールのテスト条件 $*P(*x)$ に，$\lambda *u \cdot \wedge(MEMBER(*u, v), MEMBER(*u, *w))$ のような，2変数の述語定項 $MEMBER$ から構成された式のラムダ表現を代入することができる．

上例のように，2階の単一化を用いることにより，モジュールをより広範囲なプログラムに対応させることができる．もっと複雑な例は，単一化手順の説明の後に示す．

さて，現在，一般的な2階の単一化手続きが知られているが，この手続きは一般的であるが必ずしも能率的でない．しかし，本システムのように，仕様にモジュールを直接適用する場合には，モジュールの*付きの変項にのみ代入を行うので，2階の単一化を次のように簡単化することができる．

単一化は，(3)式のようなモジュールの表現（TYPE 部は省略している）の PROC 部と展開中の仕様の手順的表現との間または，(3)式の IN および OUT 部と展開中の仕様の入出力表現との間で行われる．

```
(PROC: proc(*x, *z),
  IN: P(*x),
  OUT: Q(*x, *z),
```

OP : $op(*x, *z)$)　　(3)

いま，2個の対象を α, β とする．α は $a_1 \cdots a_k \cdots a_n$ からなる記号列で，(3)式のようなモジュールの $proc(*x, *z)$ や $P(*x)$, $Q(*x, *z)$ に含まれる述語または関数の表現とする．

同様に β は $b_1 \cdots b_k \cdots b_m$ なる記号列で，プログラムの仕様か，モジュールにいくつかの単一化代入を施した結果に含まれる述語または関数の表現とする．

a_k, b_k を α, β のなかの最左端の不一致記号とするとき，簡単化した単一化手順は次のように述べることができる．

[手順1] 簡単化した単一化手順

(1) a_k が個体変項のとき，次の代入により1階の単一化を行う．

$\theta = \{a_k \leftarrow b_k \cdots b_{k+l}\}$　　(4)

ただし $0 \leqq l \leqq m-k$ で，$b_k \cdots b_{k+l}$ は β の部分対象(式または項として意味のある一まとまりの記号列)である．

(2) a_k が α の部分対象の先頭の p 位の関数変項または述語変項のとき，次のいずれかの方法により単一化する．

(a) 射　影

a_k のタイプと a_k を先頭とする部分対象の第 i 引数のタイプが等しいとき，代入

$\theta = \{a_k \leftarrow \lambda u_1 \cdots u_p \cdot u_i \cdot\} (1 \leqq i \leqq p)$　　(5)

により a_k を先頭とする部分対象を第 i 引数に射影する．

(b) 模　倣

b_k を q 位の関数記号または述語記号とする．このとき，b_k を先頭とする β の部分対象を模倣した系列で a_k をおきかえる．

$\theta = \{a_k \leftarrow \lambda u_1 \cdots u_p \cdot b_k h_1 \cdots h_q \cdot\}$　　(6)

($h_i = f_i u_1 \cdots u_p$ で $1 \leqq i \leqq q$ かつ f_i は α にも β にも現われない p 位の関数変項または述語変項とする)

[例4] 仕　様

SEARCH (SO : $X(1..j..100, 1..10)$,
OBJ : $X(j, 1..10)$,
COND : $\wedge$ ($EQUAL(X(j, 1), a)$,
$EQUAL(X(j, 10), b)$),
GOAL : $Z(1..10)$).

($X(j, 1) = a$ で $X(j, 10) = b$ である行 j を探す)

とモジュール，

SEARCH
(SO : $*x(*n1..*i..*n2, *m1..*m2)$,
OBJ : $*x(*i, *m1..*m2)$,
COND : $\bigwedge_{*j} *P_{*j}(*x(*i, *j), *r(*j))$,
GOAL : $*z(*m1..*m2)$)

の単一化は次のように行われる．まず，$*x \leftarrow X$, $*n1 \leftarrow 1$, $*n2 \leftarrow 100$, $*m1 \leftarrow 1$, $*m2 \leftarrow 10$, $*i \leftarrow j$ なる1階の代入をモジュールに施すことにより，

SEARCH
(SO : $X(1..j..100, 1..10)$, OBJ : $X(j, 1..10)$,
COND : $\bigwedge_{*j} *P_{*j}(X(j, *j), *r(*j))$,
GOAL : $*z(1..10)$)

となる．ここで，$*P_k \leftarrow \lambda *u *v \cdot = (*u, *v)(k = 1, 10)$ および，T を真理値 $TRUE$ を表す定数として，$*P_k \leftarrow \lambda *u *v \cdot T \cdot (2 \leqq k \leqq 9)$ を施すことにより，

SEARCH
(SO : $X(1..j..100, 1..10)$, OBJ : $X(j, 1..10)$,
COND : $\wedge(=(X(j, 1), r(1)), =(X(j, 10), r(10)))$,
GOAL : $*z(1..10)$)

となり，さらに1階の代入 $*z \leftarrow Z$ を施して単一化は終了する．

[例5] 結合律を満たす2項演算子 $*f$ (たとえば $+$, $\times$, max など) を繰り返し適用して任意個の数の和や最大値などを計算する一般的手順を表す簡略表現を次のように定義する．

REPEAT ($*f, *x(*n1..*n2)$
$\equiv \#=(*z, *x(*n1))$;
FOR (COUNT : $*i$, FROM : $+(*n1, 1)$,
TO : $*n2$,
OP : $\#=(*z, *f(*z, *x(*i)))$).

上式を用いれば，$*f \leftarrow \lambda *u *v \cdot +(*u, *v)$, $*f \leftarrow \lambda *u *v \cdot max(*u, *v) \cdot$ などの代入を施すことにより，和 $REPEAT(+, X(1..n))$ や最大値 $REPEAT(max, X(1..n))$ などはその手順的表現を

$\#=(*z, X(1))$;
FOR (COUNT : $*i$, FROM : 2, TO : n
OP : $\#=(*z, +(*z, \times(*i)))$)

や

$\#=(*z, X(1))$;
FOR (COUNT : $*i$, FROM : 2, TO : n,
OP : $\#=(*z, max(*z, X(*i)))$)

のようにただちに得ることができる．

4. 詳細化と副プログラム化

SAPRE は与えられた仕様に含まれる各部分の入出力述語を調べ，値に未定義の変数が引用されている

か，また，引数の受け渡しの場合に型の不一致が生じているかを検査し，必要に応じて仕様の修正を要求する．次に，仕様に含まれる手順表現の名前や入出力表現の述語をキーとして，適用可能なモジュールを探索する．モジュールが見つかったら，副プログラム化するかどうかをユーザに問合せ，この回答に従って次に示すような詳細化あるいは副プログラム化のいずれかを行う．

4.1 詳 細 化

与えられた仕様と単一化可能なモジュールが見つかった場合には，まずモジュールに含まれるすべての変数を新しい変数によっておきかえ，このモジュールのOP部に単一化に伴う代入を施し，その結果を詳細化結果とする．もし，OP部に *?SELECT* 文がある場合には，ユーザに選択可能なOP部を提示して，そのうちの一つを選ばせ，これによって詳細化を行う．

手順的表現や入出力表現による仕様記述において，モジュールに未指定の仕様項目があれば，システムは単一化後にユーザにその指定を要求し，この入力によって詳細化を進める．

［例6］ 仕　　様

SEARCH (SO : *TABLE* (*1..i..n, 1..m*),
OBJ : *TABLE* (*i, 1..m*),
COND : =(*TABLE* (*i, 1*), *r*(*1*)),
GOAL : *z*(*1..m*)),

（上式は，「*TABLE*(*1..n, 1..m*) から第1列目が *r*(*1*) に等しい行をみつけて *z*(*1..m*) に入れる」を表している）．

が与えられた場合，表2の *SEARCH* モジュールに対して代入，

$*x \leftarrow TABLE$, $*n \leftarrow 1$, $*n2 \leftarrow n$, $*m1 \leftarrow 1$, $*m2 \leftarrow m$, $*P_1 \leftarrow \lambda *u *v \cdot =(*u, *v)\cdot$, $*P_{*j} \leftarrow \lambda *u *v \cdot T\cdot (2 \leqq *j \leqq m)$,

を行う．次に *?SELECT* 文により選択のキー *q1* の値として *KEY-SEARCH* か *GENERAL-SEARCH* を選択することをユーザに要求する．表2に示す *SEARCH* モジュールにおいて，*KEY-SEARCH* が選択された場合には，さらに *BINARY-SEARCH* か *SEQUENTIAL-SEARCH* が選択される．回答が *BINARY-SEARCH* の場合には，データがキー項目（1列目）によって *SORT* されているかをチェックし，*SORT* されていなければ *SORT* した後に *BINARY-SEARCH* を行うように仕様を詳細化する．*KEY-SEARCH* でない場合や，*SEQUENTIAL-SEARCH* が選択された場合には，*SEQUENTIAL-SEARCH* を行うように仕様を詳細化する．

BINARY-SEARCH に従って詳細化した結果は次のようになる．

IF-THEN (COND : *NOT*(*SORTED*
(OBJ : *TABLE*(*1..i..n, 1..m*),
KEY : *TABLE*(*i, 1*)),
OP : *SORT*
(OBJ : *TABLE*(*1..i..n, 1..m*),
KEY : *TABLE*(*i, 1*),
GOAL : *TABLE*(*1..i..n, 1..m*))) ;
BINARY-SEARCH
(SO : *TABLE*(*1..i..n, 1..m*),
OBJ : *TABLE*(*i, 1..m*), KEY : *TABLE*(*i, 1*),
COND : =(*TABLE*(*i, 1*), *r*(*1*)),
GOAL : *z*(*1..m*)).

4.2 副プログラム化

4.1節の方法で詳細化した結果生成されるプログラムは，procedure や subroutine を含まないプログラム構造となっている．このようなプログラムは，処理速度が大きいという利点をもつが，その反面プログラムの大きさは一般に大きく読みにくくなり，また修正や変更が困難になる．

目的プログラミング言語がもつ procedure 表現や関数表現などを用いて，コンパクトにプログラムに展開するには，上記4.1節の手順を一部変更し，次の副プログラム化の手順により，展開部を procedure 呼出しでおきかえればよい．

［手順2］ 副プログラム化の手順

1. 仕様や展開に用いるモジュールのなかに非原始的な表現が含まれているときには，これらを目的言語の procedure 呼出しに変換する．このとき，この手順的表現を見出しとするモジュールの OP 部が，procedure として展開され，すでに procedure の表に登録されているかを調べ，登録されていなければ2の手順を追加する．

2. 着目しているモジュールに単一化代入を施した後，このOP部に，このモジュールの手順的表現を変形して作成した見出し部と，モジュールの型部の情報をもとにした変数の宣言部を付ける．さらに一般言語で表されたモジュールの部分を目的言語に変換して，procedure の本体を作成する．次に，この procedure 名を procedure の表に登録する．

なお，LISP および PASCAL の場合の見出し部と

宣言部の生成規則は次の(1)および(2)である.

(1) 手順的表現から OBJ, SO, GOAL 格以外の格を除去する.

(2) 次の規則を手順的表現と型部に適用する.

{PROC : *proc* (SO : *X*, OBJ : *Y*, GOAL : *Z*)}
→(DE *proc*(*X Y Z*)　…LISP の場合

{PROC : *proc* (SO : *X*, OBJ : *Y*, GOAL : *Z*),
ENTITYTYPE :
X (REG : r_1, STRUCT : s_1,
VROLE : *INPUT*) ;
Y (REG : r_2, STRUCT : s_2,
VROLE : *LOCAL*) ;
Z (REG : r_3, STRUCT : s_3,
VROLE : *OUTPUT*) ;}
→procedure *proc*(*X* : s_1 of r_1, *Y* : s_2 of r_2,
Z : s_3 of r_3)　…PASCAL の場合

[例8] *SEARCH* のモジュールを副プログラム化するときの見出し部と宣言部は次のようになる.

(DE *SEARCH* (*X Y Z*)　…LISP の場合,
procedure
SEARCH
(*X* : *array*(*n1*..*n2*, *m1*..*m2*) of *real*,
Y : *array*(*m1*..*m2*) of *real*,
Z : *array*(*n1*..*n2*, *m1*..*m2*) of *real*),
…PASCAL の場合

副プログラム化においても, 2階の単一化を用いているので, 関数変項や述語変項をもつモジュールが使え, モジュールを柔軟に利用することができる.

5. 展開の例と実験

前章までに述べた手法を用いて, 入試の処理の仕様記述から展開を行った例を示す. 処理の仕様は次のようなものである.

「*5000* 人の受験者の受験番号 "*NUMBER*" と3科目 "*SUBJ1*", "*SUBJ2*", "*SUBJ3*" の得点の4項目が得点ファイル "*MARKFILE*" の各レコードに入っており, 各受験番号 "*NUMBER*" とこれに対応する氏名 "*NAME*" の2項目が受験者ファイル "*EXAMINEEFILE*" の各レコードに入っている. このとき,

① これらのファイルから各データを読み込み,

② 各受験者の合計得点を計算し,

③ この合計得点の順に得点表をならべ換え, 順位番号を付加し,

④ これに対応する受験者名を付けた表を作成し,

⑤ 結果を出力する.

ただし, ④において欠席者の存在を考慮に入れる.」

この仕様の処理対象は表であるから, これをシステムに入力することにより, 表の *SORT*, *COPY*, 表2に示した行の *SEARCH*, キー項目による表の合併 *JOINKEY* (*SEARCH*, *JOINT*, *ADD* を OP 部に含む) などのモジュールが提示され, これらを用いて上記の仕様から次のような仕様が作成される.

IN : GIVEN (OBJ-*FILE* :
MARKFILE(*1..5000, 1..4*)
EXAMINEEFILE(*1..5000, 1..2*)),
OUT : *EQUAL*(*MARKS*(*1..5000, 2..5*),
MARKFILE(*1..5000, 1..4*))
∧ *EQUAL*(*EXAMINEES*(*1..5000, 1..2*),
EXAMINEEFILE(*1..5000, 1..2*))) ;
①′

(IN : GIVEN (OBJ : *MARKS*(*1..5000, 3..5*),
OUT : ∀*i*∈(*1..5000*)(*MARKS*(*i, 6*)
= *REPEAT*(+, *MARKS*(*i, 3..5*)))) ; ②′

SORT (OBJ : *MARKS*(*1..5000, 2..6*),
GOAL : *MARKS*(*1..5000, 2..6*), KEY : *6*) ;
FOR (COUNT : *I*, FROM : *1*, TO : *5000*, OP :
MARKS(*I, 1*) : = *I*) ;　③′

JOINKEY (OBJ : *MARKS*(*1..5000, 1..6*) ;
PARTIC : *EXAMINEES*(*1..5000, 1..2*),
GOAL : *MARKORDER*(*1..5000, 1..7*)) ;
④′

WRITEARRAY 2
(OBJ : *MARKORDER*(*1..5000, 1..7*),
GOAL : *PAPER*, GOAL-*CNL* : *LP*)
⑤′

この仕様を SAPRE システムに入力することにより, ①′と②′の入出力表現は, それぞれ繰返しのモジュール *FOR* と例5で示した繰返し表現などにより, *READARRAY 2*

```
#-SORT(OBJ: MARKS(1..5000,1..5), KEY: MARKS(I,5),
       REL: >(Y1,Y2), MODE: BUBBLE );
#-JOINKEY(SO: MARKS(1..5000,1..5),
          PARTIC: EXAMINEES(1..5000,1..2),
          OBJ: MARKS(JJ1,1..5); EXAMINEES(JJ2,1..2),
          GO: MARKORDER(1..5000,1..6),
          COND: EQ(MARKS(JJ1,1),EXAMINEES(JJ2,1)) );
```

図1 最終的な仕様の SORT, JOINKEY の部分
Fig. 1 The parts of SORT and JOINKEY in the final specification.

```
0010 PROGRAM MAIN (MARKFILE,NAMEFILE);
0020  VAR MARKFILE: FILE OF INTEGER;
0030      NAMEFILE: FILE OF INTEGER;
0040      MARKS: ARRAY [1..5000,1..6] OF INTEGER;
0050      NAMES: ARRAY [1..5000,1..2] OF INTEGER;
0060      Y: ARRAY [1..6] OF INTEGER;
0070      MARKORDR: ARRAY [1..5000,1..7] OF INTEGER;
0080      I,J,I1,M,I2,Z,MM: INTEGER;
0090 BEGIN
0100  RESET(MARKFILE);RESET(NAMEFILE);
0110  FOR I:=1 TO 5000 DO
0120   FOR J:=2 TO 5 DO
0130    READ(MARKFILE,MARKS[I,J]);
0140  FOR I:=1 TO 5000 DO
0150   FOR J:=1 TO 2 DO
0160    READ(NAMEFILE,NAMES[I,J]);
0170  FOR I:=1 TO 5000 DO
0180   BEGIN
0190    MARKS[I,6]:=MARKS[I,3];
0200    FOR J:=4 TO 5 DO
0210     MARKS[I,6]:=MARKS[I,6]+MARKS[I,J];
0220   END;
0230  FOR I:=1 TO 5000 DO
0240   BEGIN Z:=MARKS[I,6];
0250         M:=I;
0260         FOR I1:=I+1 TO 5000 DO
0270          IF Z<MARKS[I1,6]
0280          THEN BEGIN Z:=MARKS[I1,6]; M:=I1 END;
0290         FOR J:=2 TO 6 DO Y[J]:=MARKS[M,J];
0300         FOR J:=2 TO 6 DO MARKS[M,J]:=MARKS[I,J];
0310         FOR J:=2 TO 6 DO MARKS[I,J]:=Y[J] ;
0320   END;
0330  FOR I := 1 TO 5000 DO MARKS[I,1] := I;
0340  FOR I:=1 TO 5000 DO
0350   BEGIN
0360    FOR I1:=1 TO 5000 DO
0370     IF MARKS[I,2]=NAMES[I1,1] THEN I2:=I1;
0380    FOR J:=1 TO 6 DO MARKORDR[I,J]:=MARKS[I,J];
0390    FOR J:=2 TO 2 DO MARKORDR[I,J+5]:=NAMES[I2,J]
0400   END;
0410  FOR I := 1 TO 6 DO
0420   WRITELN(MARKORDR[I,1]:4,MARKORDR[I,2]:4,MARKORDR[I,3]:4,MARKORDR[I,4]:4,
0430           MARKORDR[I,5]:4,MARKORDR[I,6]:4,MARKORDR[I,7]:4)
0440 END.
```

図 2 5章の仕様の展開結果
Fig. 2 The expansion result of the specification in ch. 5.

(OBJ : *MARKFILE*(*1..5000, 1..4*),
GOAL : *MARKS*(*1..5000, 2..5*));
READARRAY 2
(OBJ : *EXAMINEEFILE*(*1..5000, 1..2*),
GOAL : *EXAMINEES*(*1..5000, 1..2*));

および

FOR (COUNT : *i*, FROM : *1*, TO : *5000*,
OP : #=(*MARKS*(*i, 6*), *MARKS*(*i, 3*));
FOR (COUNT : *j*, FROM : *4*, TO : *5*,
OP : #=(*MARKS*(*i, 6*),
+(*MARKS*(*i, 6*),
MARKS(*i, j*)))));

と書き改められて仕様の作成を終わる．この仕様においては，*READARRAY 2* の SOurce-*ChaNeL*（入力ファイル名など）格，*JOINKEY* の *KEY* 格などのパラメータが未定義となっており，詳細化時にシステムの要求により指定される．

図 1 に仕様の一部を示す．なお図中の下線の格は，詳細化時にシステムの要求により指定された格である．**図 2** は全体の展開結果に宣言部，ファイルアクセス部を付けたものである．

6. むすび

SAPRE システムは，電子総研 LISP（システムの大きさ約 23 kW）を用いて書かれ約 10 k セルを要した．

SAPRE によるプログラムの展開の上記以外の例として，*SEARCH* や代入などの記号処理の基本モジュールを作成し，これを用いて単一化手順のプログラムや SAPRE 自身などの大まかな仕様を逐次詳細化して LISP 言語に展開した．生成に要した CPU 時間は，ACOS-6 システム 700（主記憶 6 MB）上の上記 LISP インタプリタ（セル領域を 40 k セルに設定）を用いて，単一化手順など 200 行前後のもので約 10 秒，

数十行のもので1〜2秒であった.

今後の問題として，仕様記述において詳細情報を欠く場合の処理の拡充とリンク機能の強化の問題がある．また，自然言語を用いた仕様記述の導入も本方式の一つの発展方向である.

参考文献

1) Pietrzykowshi, T.: A Complete Mechanization of Second Order Type Theory, *J. ACM*, Vol. 20, No. 2, pp. 333-365 (1973).
2) Darlington, J. L.: Automatic Synthesis of SNOBOL Programs, in Simon, J. C. (ed.), *Computer Oriented Learning Process*, pp. 443-453, Nordhoffleyden (1976).
3) Teichrow, D. and Hershey, F. A.: PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems, *IEEE Trans. Softw. Eng.*, Vol. SE-3, pp. 41-48 (1977).
4) 黒木，西田，藤田：プログラムの半自動的詳細化，情報処理学会第23回全国大会論文集，pp. 399-400 (1981).
5) 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).
6) 斎，藤田，西田：ライブラリモジュールを用いた仕様の詳細化によるプログラム生成，情報処理学会第25回全国大会論文集，pp. 423-424 (1982).
7) 西田：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法，昭和56年度科研費総合(A)（代表者田中幸吉）報告書「知識工学の基礎とその応用に関する研究」，pp. 177-188 (1982).
8) 西田：言語情報処理，第2刷，pp. 217-230，コロナ社 (1984).

(昭和58年12月21日受付)
(昭和59年3月6日採録)