プログラムの構造と論理の自動設計システム EOS/M

原田 実[†] 中村義幸[††*]

われわれは過去に解アルゴリズムの作成とコーディングを自動的に行うシステム SPACE を開発したが，SPACE を使用するには，プロセスをモジュールに分割し（構造設計），モジュールの機能仕様を条件式と実行文を用いて記述する（論理設計）など，いわゆる設計作業を行わなくてはならない．本論文では，この設計を自動化するシステム EOS/M (Equation Oriented Specification compiler for Module design) について報告する．EOS/M はバッチ型のファイル処理問題を自動設計できる．EOS/M の入力は，ファイル処理問題における実体や関連の属性項目間の関係と出力・更新要求を左辺が単一項目からなる等関係式という数式の集合で表した非手続き的仕様である．この非手続き仕様を表現するには EOS 仕様記述言語を用いる．EOS 仕様では，等関係式中のデータ項目に対しては，その項目がどの実体の属性かを指定する識別子とその項目がどのファイルに存在するのかを指定するファイル修飾子を添える．EOS/M は，等関係式をその左辺に与えられる項目の値を求める計算式と考え，計算に必要なレコード集合が同じ計算式をグループ化し，モジュールを作る．モジュール内では項目間の導出関係分析から，計算式の実行順序を決定する．さらに，モジュールに割り当てられた計算式内の項目の識別子やファイル修飾子の特性からモジュールの制御ロジックを決定し，計算式の特性からその実行条件を決定する．これらの設計結果を SPACE の条件式や処理文に変換して，最終的に正確な SPACE 仕様を生成する．

A Program Design Automation System EOS/M for Module Decomposition and Logic Design

MINORU HARADA[†] and YOSHIYUKI NAKAMURA[††*]

Almost all of the previous automatic programming systems, such as our SPACE, have set importance to the conversion from a formal specification to a program, namely to the creation of a solution algorithm and its coding in a programming language, but there are very few systems which automate module decomposition (structure design) and function specification (logic design). In this paper, we reports a system EOS/M (Equation Oriented Specification compiler for Module design) that achieves automation of both design activities. EOS/M can design batch type file processings. The input of EOS/M is a nonprocedural specification presented by a set of equations, each of which has only one item in its left hand side and expresses the relation between attribute items of entities and their relationships. In EOS specification, every item is affixed with an entity or relationship identifier and a file description. EOS/M considers each equation as the computation to give the value of its left hand side item, and collects such computations that require the same set of file records. This collection of computations becomes a module. Analysis of the derivation relationships among the items in the computations fixes the order of execution of the computaions belonging to each module. Control logics of the module are decided on the basis of the characteristics of the identifiers of the items in the computations. The timing when each computation is to be executed is determined based on its HBT type and file description. These design results are finally expressed as the action statements and condition expressions of SPACE. In a design experiment on several sorts of data processing programs, EOS/M generates the complete SPACE module specifications of high quality at the same level as the human does.

† 青山学院大学理工学部経営工学科
Department of Industrial Engineering, Faculty of Science and Engineering, Aoyama Gakuin University
†† 青山学院大学理工学研究科経営工学専攻
Department of Industrial Engineering, Graduate School of Science and Engineering, Aoyama Gakuin University
* 現在，ソニー(株)経営・技術情報システム本部業務システム部
MIS Division, Management and Engineering Information Systems Group, Sony Corporation

1. はじめに

これまでの自動プログラミング研究は，いわゆる(狭義の）プログラミングの自動化を対象にしていた．すなわち，そこでは，論理型[11]，オブジェクト指向型[2]，代数型[4]，数式[1),12)]，決定表[6),8)]，自然語[5),14)]などのさまざまな仕様モデルによる形式的仕様から，解のデータや関係を導出するアルゴリズムを構築し，これを具体的な計算式の列として生成していた．ところ

が実際のソフトウェア開発（広義にはこれ全体をプログラミングともいう）では，このプログラミング段階以前に，構造設計とか論理設計という設計段階が必要である．

構造設計は，与えられた仕様を満足するプログラムをモジュールという機能単位に階層的に分解することである．この分解は，個々のモジュールの解アルゴリズムが容易に得られるまで続けられる．一方，論理設計は，個々のモジュールに対して，それに要求されている機能を実現する論理すなわち解アルゴリズムを設計し，形式的に記述することである．

構造設計や論理設計の入力は，宣言的あるいは非手続き的に与えられた計算要件である．対象を事務処理ソフトウェアに絞ると，この計算要件を手続き的なプログラムに変換する研究として[7]，Prywes らの MODEL[12]，Ruth らの Protosystem I[13]，河野らの DSL[10]，筆者らの ARIES[5]，杉山らの KIPS[14]，などがある．しかし，MODEL では実際に要求されている計算に必要でない制御構造までも導出されてしまい最適化が難しい，Protosystem I では，複雑なデータ構造の扱いに対する考慮が欠けている．DSL では複数本ファイルに対する照合処理が難しい，ARIES や KIPS では日本文を入力仕様とするので複雑な計算条件を表現することが難しい—などさまざまな問題を含んでいる．また，このような個々の問題以外に，これらの研究が扱う問題は実問題に比べ規模や複雑性において小さく，さらに構造設計を行っていない．

本論文で報告する EOS/M (Equation Oriented Specification for Module design) は，プログラムの構造設計と論理設計を自動的に行い，形式的なプログラム仕様を生成するシステムである．対象は事務処理の中心であるファイル処理に関する実規模程度の問題である．具体的には，左辺が単一の項目からなる等関係式という数式の集合として与えられた非手続き的な要求仕様から，構造化された SPACE によるモジュール仕様群（これから COBOL プログラムへの変換は SPACE[6],[8]が自動的に行う）を自動生成する．EOS/M における自動設計の基本的手法についてはすでに報告しているので[9]，本論文においては非手続き仕様を記述する言語の詳細，システム化にあたって具体化された変換手法の詳細，および変換事例について論じる．

EOS 仕様を構成する等関係式は，実体や関連の属性項目間の関係を算術式や条件式を用いて表現している．より正確には，左辺の項目を右辺の式で定義している．EOS 仕様の各等関係式中のデータ項目に対して，それがどの実体あるいは関係の属性を表すかを指定する識別子と，どのファイルにある項目なのかを指定するファイル修飾子を添える．この識別子とファイル修飾子によって，等関係式中に与えられる各項目の値を求める元となるレコード集合を正確に決定でき，この集合が同じ計算式ごとにモジュール化する．さらに，計算式の種別やデータ項目間の導出関係分析からモジュール内での計算式の実行条件を決定する．これらの設計結果を SPACE の条件式や処理文に変換して，最終的に正確な SPACE 仕様を生成する．

2. ファイル処理問題の定式化

現在の EOS/M が自動設計できるのは，いわゆるバッチ型のファイル処理問題である．例えば，**図1**のようなものである．この問題は，二つのファイルの照合処理と二つ項目の集計（グループ）処理を含んでおり，制御論理としては簡単なものではない．

ここではファイル処理問題を定式化するとともに，自動設計の過程を説明するのに必要な諸概念を定義する（ただし，紙数の都合で諸概念の事例などの詳細については文献 9)を参照していただきたい）．EOS/M が扱うファイルとしては順編成ファイルのみを対象とする．索引編成ファイルなどのより高次のアクセス方法を持つファイルには，ファイルの順次性という制約がないために，以下の議論は容易に拡張できる．

[処理概要]
1.各社員ごとに、給与マスタの支給累計に給与を加えて更新する。
2.給与は基本給と残業手当からなる。
3.残業手当は残業代を社員ごとに集計したものである。
4.残業記録ごとに、残業代は下記の式で求める。
区分="通常"の時 残業代=単価*時間
区分="深夜"の時 残業代=単価*時間*1.5
区分="早朝"の時 残業代=単価*時間*1.2
5.当月の社員の給与明細を部合計と共に打ち出す。

[データモデル] ファイルは全て順編成ファイルである。
1.給与マスタ(新S、旧K)(部,社員の順にソート)
項目：| 部 | 社員 | 基本給 | 単価 | 支給累計 |
2.残業ファイルZ(部,社員の順にソート)
項目：| 部 | 社員 | 区分 | 時間 |
3.明細リストL(部,社員の順にソート)
明細行レコードM(社員ごとに1つ)
項目：| 部 | 社員 | 給与 |
合計行レコードG(部ごとに1つ)
項目：| 部合計 |

図 1 事例としての給与計算問題
Fig. 1 A sample salary calculation problem.

2.1 ファイル，レコード，属性項目，識別子

ファイルFは同一の型のレコードrの列であり，レコードはいくつかの項目 d*i* の値の組 〈d1, d2, …, d*n*〉であると考える．ファイルには実体や関連の属性情報が格納されている[7),9)]．ファイル処理問題は，これらのファイルから特定の実体や関連の属性値を求めたり更新することである．

ファイルFの各レコードrがどの実体や関連を表しているかを識別できる項目または項目の組Iを，実体あるいは関連の識別子と呼ぶ．また，レコードを物理的に識別する仮想の識別子を考え，@で表す．

ファイルFにおいて識別子Iの値が等しいという同値関係によって分割された個々のレコード集合をグループと呼び，特にI=値Aのものを G(F, I, A) と書く．

ファイルFにおいて，全レコードが項目 I2 で項目 I1 より先にソートされているなら，I1≦I2 とする．また一般に，任意の項目Iに対して @≦I である．なお任意の項目I，Jに対して，I≦J かつ J≦I ならI◇J とする．

2.2 ファイル処理，等関係式

ファイル処理の自動設計に対する本研究の根底となる考え方は，『ファイルLにおいて，識別子 I0（=値A）で表される実体あるいは関連の属性Dの値は，ファイル R（=L であることもある）における G(R, I0, A) 内の各レコードの，この実体や関連 I0 の別の属性 D0 や，この実体や関連と関連を持つ他の実体や関連 I1 の属性 D1 などに，関数gを適用させて求まる』ということである．ただし，ファイルRは一つのこともあれば，複数ファイルを併合した仮想ファイル（2.4節で定義する）のこともある．また，同じ実体あるいは関連を表す識別子 I0 が，ファイルによって異なるとしても，ここでは簡単化のため同一記号で表すとする．

識別子Iで表される実体や関連の属性を与える項目Dを D.I と書くと，ファイル処理は関係式

$$D.I0 = g(D0.I0, D1.I1, D2.I2, \cdots) \qquad (2.1)$$

で表せる．これを等関係式とよぶ．このように，等関係式は左辺が単一の項目からなる等式の形を取り，左辺に与えられた実体や関連の属性項目の値を右辺に与えられた算術式や条件式を用いて定義している．

等関係式(2.1)で，I0 は項目Dを属性に持つ実体あるいは関連の識別子であり，この関係式g（誤解のないとき，関数gで与えられる関係式もgで表す）の集団識別子という．これは，式gを右辺より左辺を求める計算とみた場合の計算の範囲あるいは計算対象集団を与えている．

本研究ではファイル処理を，『式(2.1)のように与えられた等関係式群を入出力データ間の満たされるべき関係とみなし，この関係を満足するデータを出力・更新する処理』と形式に定義する．

2.3 1パス性，sg 特性

プログラムが1個以上の入力ファイルからレコードを順に読み込み，あらかじめ利用者が定義した有限の作業領域を用いて，データを作業ファイルに書き出さずに処理し，結果を出力ファイルに順に書き出すとき，このプログラムは1パスであるという．EOS/Mは，中間ファイルを極力減らしてプログラムの実行効率を最適化するという観点から，1パス性を持つ（プログラムを生成する）SPACE 仕様を生成する．

1パスとして処理できない原因にはさまざまなものがある．ここでは，個々の等関係式やファイルのソート状態によって判断できる二つの場合について論じる．

第1の場合は，入出力ファイルが識別子で異なる順にソートされている順序不一致の場合である．このときは入出力ファイルのソート順を揃えるソート処理を行う必要がある．この状態を回避するには，等関係式に出てくるすべての識別子で入出力ファイルが同順にソートされている（されて出力される）と仮定する（順序条件という）．事例の給与計算の入出力は，すべてのファイルがまず部の順に次に社員の順にソートされているので，この条件は満たされている．

第2の場合は，I0 の任意の値 A0 に対する等関係式(2.1)の左辺を求めるためのグループ G (R, I0, A0) 内に，右辺にある各引数の値を求めるためのグループ G (R, I0, A0) や G (R, I1, A1) や G (R, I2, A2) が完全に包含されない場合である（ただし，A0, A1, A2, は A0 に関係する実体や関連の識別子の値とする）．含まれないときは，引数を計算するループが左辺を計算するそれらの内側に入れ子状にならないので，途中結果を中間ファイルに書き出さずに処理できない．この状態を回避するために，『個々の等関係式において，引数の識別子が≦で順に並べられ，しかも最小の識別子をJ（単位識別子という）として J≦I0, J≦Ii である』と仮定する（構造条件という）．

集団識別子と単位識別子が等しい (I0=J) とき，等関係式gを s(single) タイプといい，そうでないとき

g(group) タイプという．sタイプの等関係式は，引数の値が求まった段階で，レコード読み込みを行わずに，左辺の結果を求めることができる．sg タイプは，各等関係式の計算の実行条件を決定するために用いられる．

2.4 漸化性

等関係式(2.1)で与えられる関数関係の実際の計算について考える．G (R, I0, A0) が複数のレコードを含む場合，gは値の組 〈D0, D1, D2, …, Dn〉の列を引数としている．このことを明示化するためにgの値を決める実際の計算を行う関数を g′(G) と書く．一般に，この関数は一つの算術式で表現できない．しかし例えば，B=ΣA は初期処理 B=0 と本体処理 B=B+A で表現できる．このように，関数 g′ が(2.2)のように初期値を与えるCと，記憶域中の中間結果 g′(G′) と現在レコードrから g′(G) を求める g1 に分解できるとき，関数gが漸化性を持つという．

$$g'(G)=\begin{cases} C & G\text{が空列のとき} \\ g1(r, g'(G')) & G=[G':r]\text{のとき}\star \end{cases} \quad (2.2)$$

漸化性を持つ関数には，与えられた式のグループ内での最大：MAX，総和：SUM，先頭値：1ST などがあり，事務処理要求内の計算のほとんどがこのような関数で表すことができる．

2.5 ファイル修飾子と仮想ファイル

複数のファイルがあるとき，等関係式中の項目Dがどのファイル F に格納されているデータかを明示化するために D_F と書く．この F をファイル修飾子と呼ぶ．

事例における残業を行った社員とそうでない社員のように，ある実体の属性（例では社員の残業単価と残業時間など）が異なるファイルに記録され，『ファイルに当該レコードがあるかないか』という事実が，このレコードで表される実体の状態（例えば，残業をしたかしなかったか）を表現することがよくある．この状況は，**表1**に示すように仮想ファイルのファイル修飾子を用いて表現できる．ここで，仮想ファイル F1 & F2 は，二つのファイル F1 と F2 の各レコードを両ファイルに共通の識別子Iをキーにして，その値が同じレコード同士を連結した仮想的なレコードからなるファイルである．すなわち，識別子Iで表される実体あるいは関連のうち，ファイル F1 と F2 の両方に情報を含むレコードからなるファイルである．仮想ファイル F1 & !F2 や !F1 & F2 はそれぞれファイル F1 や F2 だけに存在する実体や関連のレコードからなるファイルである．なお，項目名は元のファイルのものを用いる．

表 1 仮想ファイル
Table 1 Virtual file.

F1	F2	表 記
有	有	F1& F2
有	無	F1&!F2
無	有	!F1& F2

⋆ [G′ : r] はレコード列 G′ にレコードrを後に繋げた列を表す．

```
①等関係式仕様
<等関係式仕様>::=+{ <一般等関係式> }.
<一般等関係式>::=<等関係式> ';'
    | IF '(' <基本条件式> ')' THEN (<等関係式> ';' | '{'<等関係式群>'}')
     [ ELSE (<等関係式>';' | '{'<等関係式群>'}')]
    | SWITCH '(' <式> ')' '{' <CASE並び> '}'
    | <ファイル修飾子> '=' <ファイル名>
    | <実体識別子> '<' <実体識別子> ';'.
<CASE並び>::=+{ <CASE文> }.
<CASE文>::=<CASE> <式> ':' ( <等関係式> ';' | '{' <等関係式群> '}' ).
<等関係式>::=<一意名> '=' <式> | WRITE '(' <レコード名> ')'
    | DELETE '(' <レコード名> ')' | REWRITE '(' <レコード名> ')' | NONE.
<レコード名>::=<一意名>.
<等関係式群>::=+{ <等関係式> ';' }.

②基本算術式
<式>::=<基本算術式> | SUM '(' <基本算術式> ')' | FIRST '(' <基本算術式> ')'
    LAST '(' <基本算術式> ')' | MAX '(' <基本算術式> ')'
    | MIN '(' <基本算術式> ')' | FOLLOW '(' <基本算術式> ')' | <定数>.
<定数>::= ALL (<表意定数>|<文字定数>)|<数字定数>
          |<文字定数>|<字類定数>|<表意定数>.
<基本算術式>::=+{+{+{<一意名>|<数字定数>|'('<基本算術式>')'|'*'|'/'|'+'|'-'|'^'}.

③基本条件式
<基本条件式>::=<単純条件式>|<否定単純条件式>
          |<組合せ条件式>|<否定組合せ条件式>|'('<基本条件式>')'.
<単純条件式>::=<比較条件式> | <字類条件式> | <正負条件式>.
<比較条件式>::=<比較条件記述項><比較演算子><比較条件記述項>.
<比較条件記述項>::=<一意名> | <指標名> | <>.
<比較演算子>::= '>' | '>=' | '=>' | '<' | '<=' | '=<' | '==' | '!='.
<字類条件式>::=<一意名> '==' <字類定数>.
<字類定数>::= NUMERIC | ALPHABETIC.
<否定単純条件式>::= '!' <単純条件式>.
<組合せ条件式>::=+{<基本条件式>: '&' | '|'}.
<否定組合せ条件式>::= '!''('<組合せ条件式>')'.

④基本構文要素
<一意名>::=<データ項目名>[OF<ファイル名>]
          ['('#(1,3)[<指標及び添字付け>':'] ')']'.'<識別子並び>.
<指標及び添字付け>::=(<指標名>|<指標付けのないデータ項目名>)
                    [('+'|'-')<整数>] | <整数>.
<識別子並び>::=+{<識別子>':'}
<識別子>::=<実体識別子>'.'<関連識別子>
<実体識別子>::=<データ名>.
<関連識別子>::='<'<実体識別子> ','< 実体識別子> '>'

⑤字句要素
<名前>::=(<日英字>[#(0,18)[['-']|<日英数字>}<日英数字>])'_'<ファイル修飾子>
<ファイル修飾子>::=+{['!']<英字>:'&'}.
<数字定数>::=[('+' | '-')] +{<数字>} ['.' +{<数字>}].
<整数>::='{<数字>}<0でない数字>'{<数字>}.
<文字定数>::=[N(A|C|H|K|N|X)](""|'') '{<クォートでない文字>|{(""|'')(""|'')}(""|'').
<表意定数>::=<表意数値定数>|SPACE|SPACES|HIGH-VALUE
          |HIGH-VALUES|LOW-VALUE|LOW-VALUES|QUOTE|QUOTES.
<表意数値定数>::= ZERO | ZEROS | ZEROES.
```

図 2 EOS 仕様の構文規則
Fig. 2 Syntax of EOS specification.

3. EOS 仕様

前章で論じた(2.1)のような関数形式の等関係式では具体的な計算の意味を表せないし，また条件指定もできない．したがって実用化の目的から，算術式や出力文や条件文を含む**図2**に示すような構文を用いて等関係式を具体的に表すことにする．この表現による仕様を EOS 仕様とよぶ．なお，これらの各式が関数gを適切に選べば(2.1)の形式を持つことは容易に確認できる．また，gはすべて漸化性を持つこともわかる．

EOS 仕様の記述方法を具体的に説明するために図1の給与計算問題に対する EOS 仕様を**図3**に示す．この問題は2本のファイルを 1：n 型で照合し，さらに出力において照合キーより大きいキーで集計するために，手作業でプログラミングする場合には通常2本の作業ファイルを用いる．後で示すが，EOS/M は作業ファイルなしの1パスで計算できる SPACE 仕様を生成する．

この仕様が表す意味は，1番目の等関係式を例に挙げると『ファイルWにおいて，識別子“社員_K”で表される実体“社員”の“給与”は，識別子“社員_K”で表される実体“社員”の“基本給”と識別子“社員_K”で表される実体“社員”の“残業手当”の和である．』となる．

4. SPACE のモジュール仕様

ここでは，EOS/M の生成対象である SPACE によるモジュール仕様の記述方法について説明する[6),8)]．

SPACE では，モジュールの仕様を**図4**に示すようなロジックテーブルに記述する．SPACE では，ファイル処理プログラムを構成するモジュールを照合，集計，検索，計算型の四つに分類し，これらの処理を行うモジュールの実行状態Sを条件式とその値の組合せ{条件式 C1＝値 v1j,…} で表す．実際，ロジックテーブルでは，『どんな状態 Sk のときどの処理 {Ai1,…} を行うか』を，『状態 Sk を表す規則列 Rk の Ci 行に値 vik を記入し，この列内の処理 Aj 行に実行指示記号Xを記入する』ことで表現する．

SPACE の最大の特徴は，ロジックテーブルの条件欄に記入して，モジュールの実行状態を簡潔に表現できる条件式が提供されていることである．

例えば，ファイルFと他のファイルのキーK上での照合処理の状態を表す MC (F；K) は，Fから入力したレコードがキーKで表される現在照合中の対象を表す状態にあることを＝Yで，そうでないことを＝Nで表す．一方，ファイルFのキーKによるグループ処理は CB(F；K) や LC(F；K) を用いて表す．CB(F；K) は，グループ処理を行うモジュールMが，他から呼ばれた初期状態か，あるいはファイルFから入力したレコードによってキーKが一定の新しいレコードグループに入った状態のとき＝H (ead) を，グループ中の各レコード(先頭や最後のレコードも同様に含んで)処理状態のとき＝B (ody) を，現在のグループを終わろうとする状態のとき＝T (ail) を値にとる．LC (F；K) は，ファイルFから入力したレコードがグループ中の先頭レコードのとき＝F(irst) を，最後のレコードのとき＝L (ast) を，中間レコードのとき＝I (ntermediate) を，グループがこのレコードのみを含むとき＝U (nique) を値にとる．これらの条件式はすべて EOF レコードを入力したときは＝E を値にとる．一方，分類処理を表す“式 D＝{式1；式2；…}”なる選択式は，式Dの値が式 i の値に等しいとき＝i を，どの式 i の値とも等しくないとき＝0 を値にとる．SPACE には，こ

```
1. 給与_W.社員_K = 基本給_K.社員_K + 残業手当_W.社員_K;
2. 残業手当_W.社員_K&Z = SUM(残業代_W.@_K&Z);
3. 残業手当_W.社員_K&!Z = 0;
   SWITCH(区分_Z.@_K&Z){
4. CASE "通常" : 残業代_W.@_K&Z = 単価_K.@_K&Z*時間_Z.@_K&Z;
5. CASE "深夜" : 残業代_W.@_K&Z = 単価_K.@_K&Z*時間_Z.@_K&Z*1.5;
6. CASE "早朝" : 残業代_W.@_K&Z = 単価_K.@_K&Z*時間_Z.@_K&Z*1.2;
   }
7. 部_S.社員_K = 部_K.社員_K;
8. 社員_S.社員_K = 社員_K.社員_K;
9. 基本給_S.社員_K = 基本給_K.社員_K;
10. 単価_S.社員_K = 単価_K.社員_K;
11. 支給累計_S.社員_K = 給与_W.社員_K + 支給累計_K.社員_K;
12. 部_L.社員_K = 部_K.社員_K;
13. 社員_L.社員_K = 社員_K.社員_K;
14. 給与_L.社員_K = 給与_W.社員_K;
15. 部合計_L.部_K = SUM(給与_W.社員_K);
16. WRITE(マスタレコード_S.社員_K);
17. WRITE(明細行レコード_L.社員_K);
18. WRITE(合計行レコード_L.部_K);
19. マスタレコード_S.社員_K = FOLLOW(支給累計_S.社員_K^部_S.社員_K
                                    ^社員_S.社員_K^基本給_S.社員_K^単価_S.社員_K);
20. 明細行レコード_L.社員_K = FOLLOW(部_L.社員_K^社員_L.社員_K
                                    ^給与_L.社員_K);
21. 合計行レコード_L.部_K = FOLLOW(部合計_L.部_K);
22. S = 新給与マスタ;
23. K = 旧給与マスタ;
24. Z = 残業ファイル;
25. L = 明細リスト;
26. W = 作業領域;
27. @<社員;
28. 社員<部;
```

ただし、
*1) 各等式の左の数字は後述の変換の説明で用いるEQUATION_Noで、EOS仕様にはない。
*2) 16, 17, 18は出力処理で、これも等関係式である。
*3) 22,23,24,25,26はファイル修飾子の宣言である。
*3) 27,28は識別子間の'<'順序を宣言する。

図3 給与計算問題に対する EOS 仕様
Fig. 3 EOS specification for the example salary calculation problem.

```
Mtx  5                    CondExp =  MC          Module->FileModifier KZ
C1  MC (旧給与マスタ;<部,社員>)                                          YYYNNNEEE
C2  MC (残業ファイル;<部,社員>)                                          YNEYNEYNE
A1  社員 OF 明細リスト := 社員 OF 旧給与マスタ .                          XXX
A2  部 OF 明細リスト := 部 OF 旧給与マスタ .                              XXX
A3  単価 OF 新給与マスタ := 単価 OF 旧給与マスタ .                        XXX
A4  基本給 OF 新給与マスタ := 基本給 OF 旧給与マスタ .                    XXX
A5  社員 OF 新給与マスタ := 社員 OF 旧給与マスタ .                        XXX
A6  部 OF 新給与マスタ := 部 OF 旧給与マスタ .                            XXX
A7  残業手当 OF 作業領域 := 0.                                            X
A8  exec                  1 .                                             X
A9  作業領域 := 基本給 OF 旧給与マスタ + 残業手当 OF 作業領域 .           XXX
A10 do                    4 .                                             XXX
A11 給与 OF 明細リスト := 給与 OF 作業領域 .                              XXX
A12 XWRITE 明細行レコード OF 明細リスト .                                 XXX
A13 支給累計 OF 新給与マスタ := 給与 OF 作業領域 + 支給累計 OF 旧給与マスタ . XXX
A14 XWRITE マスタレコード OF 新給与マスタ .                               XXX
A15 do_again.                                                             XXXXXXXX
```

```
Mtx  4                    CondExp =  LC          Module->FileModifier K
C1  LC (旧給与マスタ;<部>)                                                FILUE
A1  部合計 OF 明細リスト := 0.                                            X  X
A2  部合計 OF 明細リスト := 部合計 OF 明細リスト + 給与 OF 作業領域 .     XXXX
A3  XWRITE 合計行レコード OF 明細リスト .                                    XX
```

```
Mtx  1                    CondExp =  CB          Module->FileModifier KZ
C1  CB (旧給与マスタ;<部,社員>)                                           HHHBBBTTTEEE
C2  区分=|'通常';'深夜';'早朝'|                                           123123123123
A1  残業代 OF 作業領域 := 単価 OF 旧給与マスタ * 時間 OF 残業ファイル * 1.2.      X
A2  残業代 OF 作業領域 := 単価 OF 旧給与マスタ * 時間 OF 残業ファイル * 1.5.       X
A3  残業代 OF 作業領域 := 単価 OF 旧給与マスタ * 時間 OF 残業ファイル .          X
A4  残業手当 OF 作業領域 := 0.                                            XXX
A5  残業手当 OF 作業領域 := 残業手当 OF 作業領域 + 残業代 OF 作業領域 .   XXX
A6  do_again.                                                             XXXXXX
```

図 4 給与計算問題に対して EOS が生成した SPACE のモジュール仕様群
Fig. 4 SPACE module specifications generated by EOS/M for the sample salary calculation problem.

れら以外にも配列処理用など計九つの条件式がある．これらの条件式を条件部に併置し，これらの値の組合せでモジュールの実行状態を表現する．

給与計算問題を SPACE で記述すると，図4に示した三つのモジュールで設計できる（実はこれは後で示すように EOS/M が設計した結果である）．照合モジュール（モジュール番号5：Mtx 5）では，旧給与マスタと残業ファイルの〈部，社員〉キーにおける照合処理を条件式として MC を用いて行うことを表している．具体的には，『この照合処理は A15 の do_again 文が示すように旧給与マスタと残業ファイルが共に EOF になる（規則 EE）まで繰り返す．社員が残業をしたとき，すなわち両ファイルに照合対象レコードがあったとき（規則 YY）は，A8 において残業計算を行うモジュール1を呼び出し，このキー値を持つ残業レコードに対してその読み込みと残業の集計を行う．一方，残業をしなかったとき（規則 YN, YE）は，A7 において，残業手当に0をいれる．旧給与マスタレコードがあるとき（規則 YY, YN, YE）には，新マスタや明細リストのレコード作成（A1～A6）と出力（A11～A14）や部の合計計算と合計行出力を行うモジュール2の呼び出し（A10）を行う．』ことなどを示している．また，部合計計算モジュール（モジュール番号 4：Mtx 4）は，旧給与マスタの〈部〉キーに関する集計処理を条件式として LC を用いて表している．具体的には，『旧給与マスタ内の各社員レコードに対しては部合計への加算（A2）を，それらが各部の先頭レコードであるとき（F, U）は部合計の初期化（A1）を，最後のレコードであるとき（L, U）は部合計の出力（A3）を，それぞれ行い，これらの処理が終わると親モジュールに戻る』ことなどを示している．また，残業計算モジュール（モジュール番号1：Mtx 1）は，残業ファイルの〈部，社員〉キーに関する集計処理を，CB 式を使うことによって表している．実際，『残業ファイルからレコードを繰り返し入力するごとに，それが〈部，社員〉キー値が同じ新しい集団に入った瞬間なら（H1, H2, H3）残業手当を初期化し（A4），集団内の各レコード処理状態であるなら（B1, B2, B3）区分の値が“通常”，“深夜”，“早朝”に対応して単価と時間の積に適切な掛率を掛けた値を残業手当に加え（A1, A2, A3, A5），集団の終わりなら（T1, T2, T3, E1, E2, E3）集計値を伴って親モジュール

表 2 等関係式配列
Table 2 Equational relation matrix.

eqnNum	stateNum	sgType	eqnGen Type	eqnType	*LHSData	fileName	indexes	*groupId	*RHSExp	*unitId	ifCond or swCond			HBTType	genStr
											condType	exp1	exp2		
1	1	s	caseT	eqT	A_F	NULL	NULL	I_F	A2_F.I_F	I_F	eqeqT	A_F.I_F	1	B	A1 of F = A2 of F
2	1	s	caseT	writeT	A3_F	NULL	NULL	J_F	NULL	J_F	eqeqT	A_F.I_F	1	B	WRITE (A3 of F)
3	1	g	caseT	eqT	A4_L	NULL	NULL	K_L	A5_F.J_F	J_F	eqeqT	A_F.I_F	2	B	A4 of L = A5 of F
4	2	s	thenT	eqT	B1_L	NULL	NULL	I_F	B2_L.J_F	J_F	grteqT	B_F.I_F	3	B	B1 of L = B2 of L
5	2	s	thenT	eqT	B3_L	NULL	NULL	I_F	B4_L.J_F	J_F	grteqT	B_F.I_F	3	B	B3 of L = B4 of L
6	2	s	elseT	deleteT	B5_L	NULL	NULL	K_L	NULL	K_L	grteqT	B_F.I_F	3	B	delete (B5 of L)
7	3	s	basicT	eqT	C1_F	NULL	NULL	I_F	C2_F.J_F	J_F	NULL			B	C1 of F = C2 of F

である照合モジュールに戻る (A6)』ことなどを示している.

このように SPACE では, 構造設計はプログラムを適切な処理パターンを持つモジュールに分割することであり, 論理設計はモジュールが持つ処理パターンを表す適切な条件式を条件部に併置し, あらゆる実行状態を表す規則列を生成し, 各状態において実行すべき処理を表す行に実行指示記号を入れることである.

したがって, EOS 仕様から SPACE 仕様を作成するために行わなければならない設計作業は以下の三つである.

①等関係式 (が表す計算式) 集合のモジュール化
②モジュールの実行状態を表す条件式の決定
③等関係式が表す計算の実行条件の決定

EOS/M はこの設計作業を自動的に行い, 結果を SPACE のモジュール仕様群として生成する.

5. 構造設計の自動化

5.1 構文解析と SPACE 構文への変換

EOS/M がまず最初に行うのは, EOS 仕様を構文解析し, 結果を等関係式配列に格納することである. 例えば,

```
1. switch(A_F. I_F){
     case 1: {A1_F. I_F=A2_F. I_F;
              WRITE(A3_F. J_F);}
     case 2: A4_L. K_L=A5_F. J_F;
   }
2. if (B_F. I_F)=3)
     then {B1_L. I_F=B2_L. J_F;
           B3_L. I_F=B4_L. J_F; }
     else delete(B5_L. K_L);
3. C1_F. I_F=C2_F. J_F;
```

の三つの文からなる EOS 仕様 (先頭の数字は文番号) を考えよう. これから, **表 2** に示すような等関係式配列が生成される.

EOS/M は入力された仕様の各文を文 3 のような (2.1) で示した形式を持つ単一の等関係式に分割し, 以後の設計を行うのに必要な情報に変換して, 等関係式配列に格納する. 格納される情報には, 等式番号 (eqnNum), 元になった EOS 仕様での文番号 (stateNum), 左辺の集団識別子と右辺の単位識別子の比較による sg タイプ (sgType), 元の文が単純な等関係式か, IF 文か, SWITCH 文かなどを区別する一般等関係式タイプ (eqnGenType), 分割された個々の等関係式が代入文か, WRITE 文か, DELETE 文かなどを区別する等関係式タイプ (eqnType), 左辺のデータ項目名 (LHSData), もしあれば左辺の OF 修飾のファイル名 (fileName), もしあれば左辺の指標 (indexes), 左辺の集団識別子 (groupId), 右辺の式 (RHSExp), 右辺の単位識別子 (unitId), IF 文か SWITCH 文のときの分類条件 (ifCond or swCond), 関数形 (2.2) における初期処理か本体処理か終了処理かを表す処理タイミング (HBTType), 等関係式を SPACE 用に構文変換した処理文の文字列 (genStr). などがある.

等関係式に出てくる識別子間に構造条件が満足されれば単位識別子が存在し, 各式が表す計算が漸化性を持てば処理タイミングを決定できるので, 等関係式をこれらの配列に変換できる. この変換の際には, SPACE の処理文の構文に変換するために, 〈名前〉と〈一意名〉については, 実体識別子やファイル修飾子を取り除く.

給与計算問題の各等関係式では, @≦社員≦部なので構造条件が明らかに成り立つ, また各式は算術式や SUM 関数を用いているだけなので漸化性を持つことは容易にわかる.

等関係式の文の種類によって, SPACE 構文への変換はかなり異なる. この変換を, 等関係式の元になった EOS 文が, [A]単純な等関係式の場合 (eqnGenType=eqT), [B]SWITCH 文の場合 (eqnGenType

=caseT)，[C] IF 文の場合 (eqnGenType=thenT, elseT)，に分けて整理する．紙数の都合で変換ルールのすべてを記述することはできないが，一例を示す．

例えば，[A] の場合で，①eqnType=eqT のときで，(a)RHSExp の expType=arithT のときは，genStr は "(LHSDataName, fileName, indexes) で与えられる〈一意名〉':=' RHSExp なる〈基本算術式〉' "" であり，また HBTType=T となる．

[B]の場合は，まず同じ stateNum と swCond 内の同じ expl を持つ等関係式を探し，これらに対して swCond 内の相異なる exp 2 を求めて，"swCond 内の exp 1 から構成される〈基本算術式〉'=' '{' 最初の exp 2 から構成される〈基本算術式 1〉';' 次の exp 2 から構成される〈基本算術式 2〉';' … '}'" なる条件式を一つだけ，条件部に追加する．次に，[A]の場合にならって，各等関係式の eqnTypc によって，計算式を処理部に記述する．

[C] の場合は，まず，同じ stateNum を持つ eqnGenType=thenT あるいは elseT の等関係式を探し，ifCond から構成される〈基本条件式〉を一つだけ，条件部に追加する．次に，[A] にならって，各等関係式の eqnType によって，計算式を処理部に記述する．

5.2 導出順序の決定

データの導出関係からくる計算式間の実行順序（導出順序と呼ぶ）→を決定する．一般に二つの等関係式の引数の間に，A=f(B1, …, B, …, Bm) かつ B=g(C1, …, C, …, Cn)なる関係があれば，Bを定義する g の計算が終了するまで f の計算を開始できないので，g→f とする．また等関係式の expType が sumT, maxT, minT の場合，この等関係式の初期処理 (HBTType=H) →本体処理 (HBTType=B) とする．

EOS 仕様内の等関係式による項目関係が循環定義を含まないためには，各等関係式を頂点に→順序を有向辺にしたグラフが位相ソートできなければならない（無循環条件という）．

給与計算問題に対する EOS 仕様は無循環条件を満たしており，その→順序は，等関係式を番号で表すと，{7〜10, 11}→19→16, 12〜14→20→17, 1→14, 15→21→18, 4〜6→2→1, 3→1→{11, 15}, 3→22, 15→23 となる．

5.3 計算式のグループ化によるモジュール作成

集団識別子がファイル修飾子まで含んで同じ等関係式は，同一のレコードグループに対する計算である．したがって，これらを処理要素とするモジュールを作成する（この際，◇関係にある識別子はどちらかに統一する）．

この集団識別子をモジュールのモジュール識別子（誤解のないときは，単に識別子）という．また処理要素となる等関係式が一つでも g タイプであればモジュールの sg タイプを g，そうでなければ s と定義する．事例に対しては，**表 3** のようなモジュールが作成される．なお，日本語のモジュール名は説明のために添えたもので，EOS/M が与える名前は単なる番号のみである．

5.4 モジュールの階層化

同じファイル修飾子Fを伴う識別子を持つモジュールを頂点，モジュール識別子が≦関係にありその間に同様の他のモジュールが存在しないモジュールAとBの間に有向辺（A→Bで表す）を持つ，モジュールグラフ MG(F) を作成する．

次に，各グラフをモジュール識別子の≦順序で位相ソートし，最小の識別子を minI(F) とする（階層条件という）．ソートできなければ，個々の等関係式に対する構造条件の場合と同じように，各モジュールで行われる計算が全体として入れ子状態にならず 1 パスで計算できないので除外する．

MG(F) において，ファイル修飾子が複数ファイルを含むなら (F=F1^…^Fm で m≧2)，これらのファイルからレコードを識別子 minI(F) で同期を取りながら入力する照合処理を行うために，minI(F) を識別子に持つ照合モジュール MC(F) を新たに作成し，MG(F) に追加する．このとき，MG(F) 内のモジュールNに対して，その識別子を I とすると，minI(F)≦I または I≦minI(F) なら，辺 MC(F)➡N または辺

表 3 事例に対して生成されたモジュールの識別子と sg タイプ
Table 3 Identifier and sg type of the modules generated for the sample problem.

モジュール番号：仮の名	所有する等式の番号	識別子	sg タイプ
1：転記出力	1, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 19, 20	社員_K	s
2：残業計算	2, 22	社員_K&Z	g
3：残業なし計算	3	社員_K&!Z	s
4：残業代計算	4, 5, 6	@_K&Z	s
5：部合計計算	15, 23, 18, 21	部_K	g

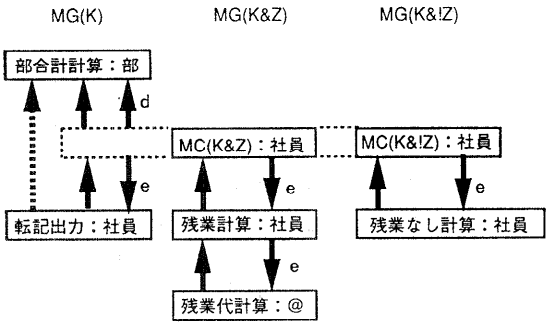

図 5 事例に対するモジュールグラフ
Fig. 5 Module graph generated for the sample problem.

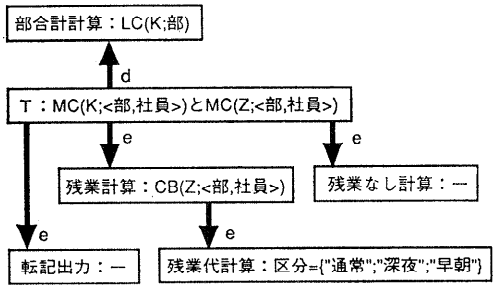

図 6 事例に対するモジュール階層と条件式
Fig. 6 Module hierarchy and condition expressions generated for the sample problem.

N➡MC(F) を（I◇minI(F) の時は辺 N➡MC(F) を）追加し，逆にこれによって推移的に順序がつく頂点間の辺は削除する．

事例では，**図5**に示す三つのモジュールグラフ MG(K) と MG(K & Z) と MG(K & !Z) が生成され，@≦社員≦部を考慮に入れると，これらはいずれも階層条件を満足することがわかる．

さらに，モジュール間の呼び出し順序を決定するために，以下のグラフの統合と変形を行う．

1) 全モジュールグラフを併合したグラフ MMG を作成する．このとき，（ファイル修飾子の !F は F と同一視して）同一の識別子を持つ照合モジュール MC(F) を融合する．
2) MMG が連結グラフになるとき，照合モジュールを表す頂点をMラベル付けする．M頂点の中で最大の識別子を持つ頂点が，最初に実行されるトップモジュール（Tで表す）となる．M頂点がないときは，各 MG(F)のFが単独ファイルなので，最小の識別子（ないときは @F を識別子に持つモジュールを追加する）を持つモジュールのラベルをCにし，これをトップモジュールとする．
3) MMG 内で，Tから葉に至る有向路上の頂点をLラベル付けし，辺➡を辺➡dに換える．このラベル付けは照合モジュールを中心に，それより大きいキーのグループ処理モジュールは LC 条件式を用いてコルーチンとして呼び出すことを意味している．
4) MMG 内で葉からTに至る有向路上のM頂点以外の頂点をCラベル付けし，C頂点を始点に持つ辺➡を逆向きの辺➡eに換え，M頂点を始点に持つ辺➡を逆向きの辺➡dに換える．この変形は照合モジュールを中心に，それより小さいキーに対するグループ処理モジュールは CB 条件式を用いてサブルーチンとして設計することを意味している．

このようにして，作成されたモジュール群がTラベルモジュールを頂点にして階層化され，構造設計が完了する．なお，状況によっては連結グラフにならなかったり位相ソートされないときもある（このときの対処の詳細は文献9)）．

事例では，図5のモジュールグラフから，連結グラフになりまた位相ソートもされるので，**図6**に示すような階層化された六つのモジュールが作成される．

6. 論理設計の自動化

論理設計で行うことは，構造設計の結果作成された各モジュールの制御論理の決定と割り当てられた計算式の実行条件の決定を行うことである．

6.1 モジュールの制御論理の決定

SPACE 仕様においては，先に述べたように，モジュールの制御論理は条件欄に記入する条件式の組合せとして表される．MMG の各モジュールに対する条件式は**表4**に示す規則に従って決定される．ここで，モジュール識別子を I_F とすると，Fにおいて識別子Iに≦順でそれより大きい項目をすべて逆に繋げた識別子を，識別子Iに対応するキーと呼び，key(I_F) と書く．表4は，いわば SPACE の設計者の経験的知識を整理したものであり，プログラム設計用のエキスパートシステムの知識ベースともいえる．

この方法で決定される条件式は，モジュールが行うレコードの入出力制御を決定するものだけである．もしモジュールに割り当てられた等関係式内に EOS 仕様の switch 文から派生した eqnCondType=caseTなる等関係式があれば，データ項目の値による分類を行う条件式を追加する．具体的には，まず，同じ stateNum と swCond 内の同じ expl を持つ等関係式を

表 4 モジュールの条件式と追加命令の決定法
Table 4 The rules to decide condition expressions of a module and action statements added to the module.

トップか	sg タイプ	ラベル	条　件　式	追加命令
はい	—	M	n個のMC(Fi; key(I_Fi))	do_again
はい	—	C	n個のCB(Fi; key(I_Fi))	do_again
いいえ	—	M	n個のMC(Fi; key(I_Fi))	xread Fi
いいえ	s	L, C	なし (—で表す)	なし
いいえ	g	L	LC (Fi; key(I_Fi))	do_again
いいえ	g	C	CB (Fi; key(I_Fi))	do_again

探し，これらに対して swCond 内の相異なる exp 2 を求めて，“swCond 内の expl から構成される〈基本算術式〉‘=’‘{’ swCond 内の最初の exp 2 から構成される〈基本算術式 1〉‘;’ swCond 内の次の exp 2 から構成される〈基本算術式 2〉‘;’…‘}’” なる条件式を，条件部に追加する．

結局事例に対しては，図 6 に示したように各モジュールの条件式が決定される．

6.2 処理命令の追加

モジュール内で計算を繰り返し行うか行わないかは，モジュールがトップモジュールであるかどうか，モジュールに割り当てられた計算が s タイプか g タイプか，などによって決まる．結局これは，表 4 のように整理され，必要に応じて “do_again” を処理要素に追加することになる．

さらに，モジュールグラフにおけるモジュール A からモジュール B への辺 A➡B ごとに，それが A➡dB ならサブコルーチン呼び出し “do B” を，A➡eB ならサブルーチン呼び出し “exec B” を，モジュール A の処理要素に，B を呼び出す命令として，追加する．

6.3 モジュールの統合

s タイプのモジュールでは，その引数の値が確定した段階で，レコードの読み込みを行わずに計算を完了できる．したがって，このモジュールを呼び出す親モジュール内に，その呼び出し処理の代わりに，このモジュールの処理要素を直接展開することにする．

このモジュールの統合においては，条件式は単に条件欄において併合するだけでよい．事例では図 6 において，{転記出力，残業なし計算} を T 内に，残業代計算を残業計算内に展開する．

6.4 計算条件の決定

最終段階として，モジュールに割り当てられた計算式の実行条件を決める．

まず，ロジックテーブルの条件部にこれまでに求まった条件式を記入する．次に，これらの条件式の相異なる値の組合せであるすべての規則列を生成する．一方，処理部には，モジュールに割り当てられたすべての計算式（モジュールの呼び出し命令，do_again, xread などの追加処理要素も含む）を，これらに対する導出順序→の順に記入する．ただし，モジュールの呼び出し命令については，呼び出されるモジュールの処理要素との導出順序を考慮する．また，do_again と xread は最後に記入する．

最後にこれら各計算式が，モジュールのどんな状態のときに実行されるかを決定する．この決定ルールを，**表 5** と**表 6** に示した．表 5 は条件部に記入された条件式が入出力制御に関する MC, CB, LC の場合を表し，表 6 は分類条件の場合を表している．具体的にはこれらの表は，『処理部に記入された計算式が各表の右欄に示すどの特徴を持つかによって，条件部に記

表 5 モジュール内の計算式の実行順序の決定方法 1
Table 5 The decision rule 1 of the execution condition of each computation in a module.

条 件 式	条件値	実行すべき計算式
MC (F; K)	Y	ファイル修飾子*1がFを含む
MC (F; K)	Y	xread F
MC (F; K)	Y	do_again
MC (F; K)	N	ファイル修飾子が $\overline{F}$ を含む
MC (F; K)	N	do_again
MC (F; K)	E	NOP
CB (F; K)	H	初期処理*3
CB (F; K)	H	do_again
CB (F; K)	B	本体処理*4
CB (F; K)	B	do_again
CB (F; K)	T	終了処理*5
CB (F; K)	T*2	do_again
LC (F; K)	F	初期処理
LC (F; K)	F	本体処理
LC (F; K)	I	本体処理
LC (F; K)	L	本体処理
LC (F; K)	L	終了処理
LC (F; K)	U	初期処理
LC (F; K)	U	本体処理
LC (F; K)	U	終了処理

*1) 式 g の集団識別子，あるいは呼び出し命令の呼び出し先モジュール識別子のファイル修飾子．
*2) これがトップモジュールの時のみ．
*3) 初期処理とは，HBTType＝H なる等関係式．
*4) 本体処理とは，HBTType＝B なる等関係式．
*5) 終了処理とは，HBTType＝T なる等関係式．

表 6 モジュール内の計算式の実行条件の決定方法 2
Table 6 The decision rule 2 of the execution condition of cach computation in a module.

条件式	条件値	実行すべき計算式
S={…; Si; …}	i	S=Si に対応する計算式*1
論理式C	Y	C=Y に対応する計算式*2
論理式C	N	C=N に対応する計算式*3

*1) eqnGenType=caseT で，swCond 内の i 番目の exp 2 を持つ式.
*2) expGenType=thenT なる式.
*3) expGenType=elseT なる式.

入された各条件式がこの表の左欄に示すタイプを持つ行の中央欄に記入された値をとる規則列に，実行指示記号Xを入れる』ことを指示している.

この結果事例に対しては，先に図4に示したSPACEの三つのモジュール仕様が生成される．この仕様は4章で説明したように確かに図1に示した給与計算を行う．この設計結果は人手によって設計した結果と，読みやすさのために計算式のモジュール内での位置が異なる点などを除いて，論理的には全く同じものであった.

特にレコードの入力制御に関しては，照合モジュールがトップモジュールになり〈部，社員〉キーによる照合処理を繰返し行う．これから呼出される部合計計算モジュールは，旧給与マスタの〈部〉キーに関する集計処理を行うが，親モジュールが〈部，社員〉キーに関する繰返しを行い，しかも〈部〉キーの方が〈部，社員〉キーより大きい集団を識別しているので，レコードごとに状態を把握できる LC 式を条件式として用いるように設計されている．さらに処理として do_again を使わないで，モジュール内での繰返しを行わないサブコルーチンになるように設計されている．残業計算モジュールは〈部，社員〉キーによる繰返しなので，CB 式を条件式に用いた設計となっている．また，内部で do_again によって繰返しを行いサブルーチンとして呼び出されるように設計されている．このように，サブコルーチンを巧く使った設計が生成されるので，中間ファイルを用いず1パスで処理され実行時間は短くなる．これは言い換えれば，EOS/M を使えば高度な設計テクニックを用いた SPACE 仕様が自動的に生成されることを示している.

7. おわりに

EOS/M におけるプログラム自動設計の基本方針，
①同一実体や同一関連に対する計算は同一モジュールで行う
②データの導出順序から実行順序を決定する
③SPACE の設計知識をルール化して，モジュールの階層化と計算式の実行条件を決定する
に従って，データ間の等関係式の集合から構造化されたプログラム仕様を自動生成することができた.

論文で示した給与計算以外にも在庫管理や分類計算などの問題を表す約5〜25行の等関係式群を用いて生成実験を行った．いずれにおいても，人手による設計と同程度の質の高い SPACE のモジュール仕様が生成された．この結果，ファイル処理に関して，プログラムの構造設計と論理設計を自動的に行うという目標は達成された.

今後の課題としては，次の二つがあげられる.
①SPACE の設計インタフェースに EOS 仕様の編集ウインドウを追加する．また，EOS/M の設計結果を SPACE のロジックテーブルやモジュール階層図の仕様ファイルの形式に合せて出力する．これによって，EOS/M と SPACE を完全に結合することができ，一貫した自動開発を実現できる.
②EOS/M の構文を拡張し，データ項目の実体や関連による“.”修飾を現在の1段のものから多段のものに拡張する．これによって，データ間の関連を2ファイル以上に跨がるものまで扱えるようになる．これは，ソフトウェア設計におけるプロセス設計にあたり，中間ファイルの自動設計を含むもので，今後の大きな課題であり，自動化のために現在 EOS/P というシステムを開発中である.

謝辞 本システムの開発に協力してくれた原田研究室の神山幸一（現(株)東芝），浅見仲美（現(株)東芝），前島康伯（現(株)日立製作所）の諸氏に感謝する.

参考文献

1) Barstow, D.: Domain-Specific Automatic Programming, *IEEE* Trans. Softw. Eng., Vol. SE-11, No. 11, pp. 1321-1336 (1985).
2) Balzer, R.: Operational Specification as the Basis for Rapid Prototyping, *ACM SIGSOFT, Software Engineering Notes*, Vol. 7, No. 5, pp. 3-16 (1982).
3) Chen, P. P.: The Entity-Relationship Model—Toward a Unified View of Data, *ACM Trans.*

on Database Systems. Vol. 1, No. 1, pp. 9-36 (1976).
4) Darlington, J.: An Experimental Program Transformation and Synthesis System, *Artif. Intell.*, Vol. 16, pp. 1-46 (1981).
5) 原田 実, 篠原靖志: 部品合成によるプログラム自動生成システム ARIES/I, 情報処理学会論文誌, Vol. 27, No. 4, pp. 417-424 (1986).
6) Harada, M.: Specification Compiler: SPACE, *Proc. of IEEE COMPSAC '87, Tokyo*, pp. 171-180 (1987).
7) 原田 実: 事務処理分野における自動プログラミング, 情報処理, Vol. 28, No. 10, pp. 1378-1397 (1987).
8) 原田 実: COBOL プログラム自動生成システム SPACE による仕様の視覚化と抽象化, 信学論, Vol. J 71-D, No. 7, pp. 2555-2562 (1988).
9) 原田 実, 二方厚志: 非手続き的仕様から手続き的仕様への変換—SPACE 仕様への変換手法—, 信学論, Vol. J 72-D-1, No. 4, pp. 262-271 (1989).
10) 河野史男, ほか: データに着目した仕様を入力とする COBOL プログラム生成システム DSL の開発, 日立評論, Vol. 65, No. 7, pp. 53-58 (1983).
11) Manna, Z. and Waldinger, R.: A Deductive Approach to Program Synthesis, *ACM TOPLAS*, Vol. 2, pp. 90-121 (1980).
12) Prywes, N. S. and Pnueli, A.: Compilation of Non-procedural Specifications into Computer Programs, *IEEE Trans. Softw. Eng.*, Vol. SE-9, No. 3, pp. 267-279 (1983).
13) Ruth, G.: ProtosystemI: An Automatic Programming System Prototype, *Proc. of the NCC, Anaheim, Calif., AFIPS 47*, pp. 675-681 (1978).
14) 杉山健司, ほか: 対話型自然言語プログラミングシステムの試作, 信学論, Vol. J 67-D, No. 3, pp. 297-304 (1984).

(平成4年7月 1 日受付)
(平成5年6月17日採録)

原田 実 (正会員)

1975年東京大学理学部物理学科卒業. 1980年東京大学理学系大学院博士課程修了. 理学博士. 同年(財)電力中央研究所担当研究員. 1989年青山学院大学理工学部経営工学科助教授. 専門は自動プログラミング, CASE, オブジェクト指向分析／設計論, エキスパートシステム, 自律ロボットなど. 1986年(財)電力中央研究所経済研究所所長賞. 1992年人工知能学会全国大会優秀論文賞. 著書として,「ソフトウェアの構造化設計法」(日本コンピュータ協会),「自動プログラミングハンドブック」「CASE のすべて」「知的プログラミング」(オーム社)など.

中村 義幸

1969年生. 1991年青山学院大学理工学部経営工学科卒業. 1993年同大学院理工学研究科経営工学専攻修士課程修了. 1993年ソニー(株)入社. 現在, 同社経営技術情報システム本部業務システム部企画推進課に所属.