ソフトウェア工学 101−11
（1994. 11. 18）

非手続き仕様からのプロセス設計の自動化システムEOS/P
An automatic process design system EOS/P

原田 実*, 西村淳一**
Minoru Harada*, Junichi Nishimura**

要旨

ソフトウェア開発過程の中でも中流工程であるプロセス設計の自動化は重要であるにもかかわらず非常に遅れている．我々は対象をファイル処理問題に限定し，データの依存関係と構造に着目して，与えられた非手続き的要求からそれを満足するプロセスの流れを設計する理論を確立し，これに基づく自動設計システムEOS/Pを開発した．EOS/Pの入力は，実体や関連の属性間の関係と出力／更新要求を表した等関係式という数式の集合である．ここでは属性はそれが属する実体や関連の識別子で修飾され，この修飾は一段の時もあれば，関連を経由して多段になることもある．EOS/Pは不定長の計算結果をファイルに掃き出さずに処理できる仕様を生成することを目的にする．したがってEOS/Pは，まず多段修飾されている属性に対して，複数の関連ファイルを経由してこれらの属性を修飾する識別子と元になる属性を共に含む中間ファイルを作成し，これを用いて与えられた多段の等関係式をそれと等価な一段修飾の等関係式に変換する一連のプロセスを生成する．さらに，順序不一致や構造不一致がある場合は適切なソートプロセスや照合プロセスを追加する．最終的に得られた各プロセスの等関係式集合は，すでに発表済みのEOS/MとSPACEがCOBOLコードへと変換する．

Abstract

There are very few systems to automate process design in spite of its importance in the middle phase of software development. We established a theory to design a process flow depending on the functional dependency between data item and the structure of data, and then developed the automatic process design system EOS/P. EOS/P inputs a non-procedural specification of a business data processing, where the relationship among attributes of entities and associations are represented by equations. EOP/P analyses such a specification on the basis of the proposed theory, and then decomposes it into subspecifications, each of which represents the function of a single process and can be transformed automatically into COBOL programs by our previously proposed automatic programming systems EOS/M and SPACE.

1. はじめに

本研究の対象は，事務処理問題，特にその主流であるファイル処理問題に関する設計の自動化である．ファイル処理問題（以降ジョブとよぶ）とは，いくつかのファイルに格納されたデータから指定した関係を満足するデータを持つファイルを生成したり，既存のファイル内のデータの値を指定した関係を満足するように更新することである．

ファイル処理問題におけるプロセス設計とは，ジョブに与えられた仕様を満足する処理を一連のプロセスと呼ばれるファイル処理とその間を流れる中間ファイルとして設計することである．ここで，1プロセスは1プログラムからなる実行単位であり，処理ロジックを簡単にするためにプロセス内では不定長の計算結果をファイルに掃き出さずに処理できるように設計するのが普通である．

プロセス設計の支援系としては，頻度，開始時期，処理時間などが同じ処理をまとめてプロセスとして生成するMBA社のPRIDE-ASDMなどがあり，モジュール設計の支援系としては構造化分析に基づいてデータフロー図からモジュール階層を抽出するNTTの黒木らのSoftDA[9]などがある．しかし，これらはあくまでも支援系であり自動化システムではない．一方，自動化システムとしては，筆者らによるARIES[4]やSchlumberger-Doll研究所のBarstowらによるΦNIX[1]などがある．ARIESは日本語要求を受け付け，そこに表された項目間の従属関係を分析し，結果を最終的にHYPERCOBOLというデータフロー型言語によるプログラムとして生成する．しかしARIESでは，いわゆる入出力データ間に順序不一致や構造不一致[8]がある場合は処理できないし，また与える要求記述において条件分岐を伴う複雑な計算を指定できないなどの制約があった．ΦNIXは，非形式的で不完全なプログラム仕様からPASCALプログラムを生成する．この過程で，領域知識を用いて仕様を完全で形式的なものに変換し，それをプログラミング知識を用いて繰り返しや条件分岐を含むプログラムに変換する．しかしΦNIXの主題は領域知識の利用であり，大規模事務計算に必要なプログラム設計やモジュール設計の自動化は全く行われていない．

本研究の第1の目的は，対象をバッチ型のファイル処理問題に限定し，ファイルおよびファイル処理を定式化し，この定式化に基づいた等関係式仕様という仕様記述言語を提案することである．第2の目的は，等関係式仕様で記述されたジョブ要求を満足する一連のプロセスとその間を流れる中間ファイルを設計する理論を確立することである．第3の目的は，プロセス設計の自動化システムEOS/P(Equation Oriented Specification compiler for Process Design)を開発し，我々の理論の有効性を実証することである．

EOS/Pの入力である等関係式仕様では，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・更新要求を等関係式という数式の集合で表わす．等関係式では，左辺は単一の項目からなり，右辺はそれを定義する項目からなる算術式や条件式である．ここで項目は"."を用いてそれが属する実体や関連の識別子で修飾される．この修飾は一段の時もあ

*青山学院大学 理工学部 経営工学科
**青山学院大学 理工学研究科 経営工学専攻

れば，関連を経由して多段になることもある．さらに項目は，"_"を用いてそれが存在するファイルの識別子で修飾されている．EOSにおいては，ジョブを構成するプロセスを，その制御ロジックを簡単にするために，不定長の計算結果をファイルに掃き出さずに処理できるように設計する．このためにEOS/Pは，まずジョブに対する等関係式仕様内の多段修飾を分析し，これを計算対象のレコード集合のしぼり込みと考え，非修飾項目を修飾項目に併合した中間ファイルを次々に作成する一連の照合プロセスを生成する．この際，順序不一致や構造不一致がある場合は適切なソートプロセスや照合プロセスを追加する．この結果，初期の多段の等関係式仕様は，最終的に得られた中間ファイルを用いたそれと等価な一段修飾の等関係式集合に変換され，1プロセスとして実行可能となる．なお，EOS/Pが生成した一段修飾のプロセス仕様は，EOS/M[6,7]によって同一対象に対する計算式を集めたモジュールに分割され，さらに計算式の計算条件が決定され表形式で記述したモジュール仕様群に変換される．最後にこれらモジュール仕様はSPACE[5]によって,指定された計算条件下の計算を正しく行うアルゴリズムを展開してCOBOLプログラムに変換される．

以下では，2節においてファイルとファイル処理を形式的に定義し，さらにプロセスが1パス性を持つ条件を明らかにする．3節においては等関係式仕様記述言語を定義する．4節においては，多段修飾を含む一般の等関係式仕様から1パス性を持つプロセスへの分解規則を明らかにし，EOS/Pの有効性を事例を用いて説明する．5節は結論である．

2． ファイル処理の定式化

事務処理では必ずその前提として，用いられるデータがどの様にファイルに格納されているかがあらかじめ定められている．これを図1(a)の様に表現したものをデータモデルと呼ぶ．この例では，部と社員と掛率という3つの実体の情報が幾つかの関連ファイルを介して結合している．例えば，実体"社員"は実体ファイル"給与マスタ"で表され，関連"所属する"は関連ファイル"所属ファイル"で表されている．なお，関連ファイルとは，例えば"所属する"の場合，部に対してその部に所属する社員がn人いる時，これらの実体の識別子である部Noと社員Noを持つレコードを含むことによりこの1:n関係を表すファイルである．

ここではこのようなファイル処理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．なお，EOS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．索引編成ファイルやデータベースなどのより高次のアクセス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．

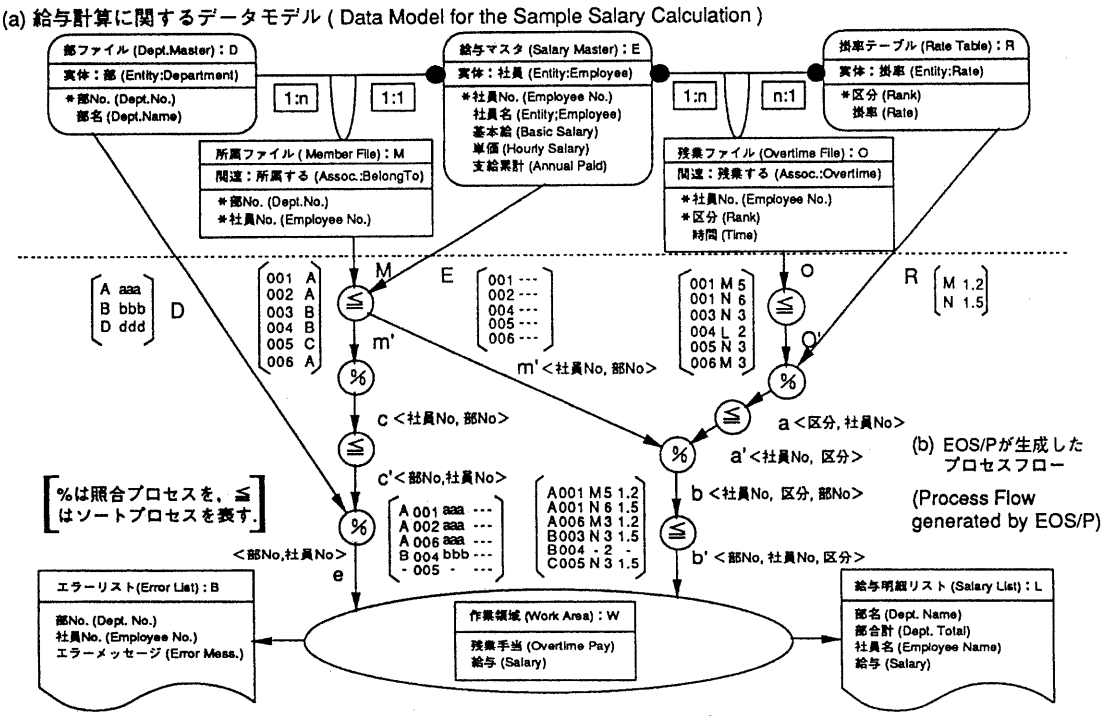

図1　給与計算例題の外部ファイルとそこからEOS/Pが生成したプロセスフロー
Fig. 1 Input/Output files of a sample salary calculation problem and the generated intermediate files.

2．1　ファイル，レコード，グループ

ファイルには，実体関連モデル[2]で言うところのいわゆる実体†1や関連の属性情報が格納されているとする．ファイル処理とは，これらのファイルから特定の実体の属性値を求めて，帳票を出力したりファイルを更新することである．一般に，xiを特定のドメインDiの要素として†2，レコードfは値xiの組<x1,x2,…,xn>†3であり，属性項目XiはレコードfからドメインD_iへの写像X_i:f→D_iであり，ファイルFは同一の型のレコードfのリスト{f}†4である．ファイルFの各レコードfがどの実体を表しているかを識別できる項目Iを実体の識別子と呼ぶ．またどの関連を表しているかを識別できる項目の組<I1,...,In>を関連の識別子と呼ぶ．この時，実体Iあるいは関連<I1,...,In>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，@で表わす．

事例として表1に示した部社員ファイルは，内容的には社員を表す実体ファイルであるが，部NOを含むので誰がどの部に属するかを表す関連ファイルとも考えることができる．なお，社員103のように複数の部に属するものもあるので，この関連は多対多である．また個々の関連を識別するのは<部NO,社員NO>と@である．

部NO	社員NO	社員名	基本給	単価	@
A	103	山本一郎	290000	2500	001
A	105	川端二郎	280000	2300	002
B	106	石坂三郎	270000	2300	003
C	120	赤坂四郎	290000	2500	004
C	103	山本一郎	290000	2500	005

表1　事例としての部社員ファイルM
Table 1 A sample Department-Employee File.

項目Dがファイルに属する項目であることをEOSではD_Fと表すが，論文中では字数を詰めるために添え字を用いてD_Fと表す．この時，添え字Fを項目Dのファイル修飾子と呼ぶ．この記法を用いて，ファイルFが識別子I2で識別子I1より先にソートされているなら，$I1_F$<$I2_F$とする．また一般に，任意の識別子I_F,I'_Fに対して$@_F$≦I_F,<I,I'>$_F$<I_F,<I,I'>$_F$<I'_Fである．表1では，$@_M$<<部NO,社員NO>$_M$<社員NO_M<部NO_Mである．

ファイルFにおいて項目X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リストをグループと呼び，特にX(f)=xのものをgroup_F(X,x)と書く．

group_F(X,x)={f| ∃f∈F,X(f)=x}†5…(1)

2．2　識別子による属性の修飾

ファイルFが実体を表し，その識別子がI_Fで，項目X_Fが実体I_Fの属性である時，実体I_Fの属性X_Fの値リストを$X_F.I_F$と書く．この時，項目X_Fは識別子I_Fで修飾されているという．しかし，1つの実体の属性が複数のファイルに存在する時や，新しいファイルを作成する時には，属性のファイル修飾子とそれを修飾する識別子のファイル修飾子とが異なることもある．従って，一般的には，$X_F.I_G$はファイルG内の各レコードgの識別子Iの各値iに対するファイルF内のグループgroup_F(I,i)内の各レコードfの項目Xの値リストをiの昇順に結合したリストとして定義される．この様な"."1つによる，実体Iと属性Xの修飾を一段修飾という．

$X_F.I_G$={X(f)|∃g∈G,∃f∈group_F(I,I(g))}…(2)

例えば，部NO_M.社員NO_M={A,C,A,B,C}となる．明らかに(2)式は式(1)より，次のようにも書くことができる．

$X_F.I_G$={X(f)|∃g∈G,∃f∈F,I(f)=I(g)}…(3)

この記法をさらに拡張して，ファイルFとGが実体Iの情報を含んでいたとして，Gで特徴付けられる性質を持たない実体Iの属性X_Fの値リストを$X_F.I_{IG}$と書く．具体的にはファイルG内のレコードが持つ識別子Iの値以外の値i'を持つファイルF内の各レコードfの項目Xの値をi'の昇順に並べたリストとして定義される．

$X_F.I_{IG}$={X(f)|∀g∈G,∃f∈F,I(f)≠I(g)}…(4)

しかし，$X_{IF}.I_G$={X(f)|∃g∈G,∀f∈F,I(f)≠I(g)} は意味がない．なぜならレコードfの量化記号が∀なので集合を形成できないからである．なお，これらの記法はGが関連を表すファイルの時も同様に拡張できる．この時は識別子は<I1,I2>など実体識別子のペアであり，X_F.<I1,I2>$_G$などと書く．

この"."修飾は関連ファイルを介在することによって多段にすることができる．すなわち，出力／更新の要求単位となる実体I1を表すファイルGと属性Xを持つ実体I0を表すファイルF間をいくつかの関連ファイルRによって結合することによって，属性Xが実体I1とどの様な従属関係にあるかを"."修飾を複数回用いて指定できる．例えば，実体I1とRで表される関連を持つ実体I0の属性は$X_F.I0_G.<I1,I0>_R.I1_S$と表される．形式的には，属性Xの値を求める元になるレコードリストは，実体I1の各値i1に対して識別子<I1,I0>を持つ関連ファイルR内のI1(r)=i1なるレコードrと実体ファイルG内のI0(g)=I0(r)なるレコードgを経由して関係しているグループgroup_F(I0,I0(g))で与える．すなわち，

$X_F.I0_G.<I1,I0>_R.I1_S$={X(f)|∃s∈S,∃r∈R,∃g∈G,∃f∈F,I1(r)=I1(s),I0(g)=I0(r),I0(f)=I0(g)}…(5)

また，経由する途中ファイルを実体ファイルにすることもできるし，修飾元Uや修飾先Gを実体から関連に拡張することも同様にできる．例えば，$X_F.I0_G.<I1,I0>_R.I1_S.<I2,I1>_{IT}.I1_U$={X(f)|∃u∈U,∀t∈T,∃s∈S,∃r∈R,∃g∈G,∃f∈F,I1(t)≠I1(u),I1(s)=I1(u),I1(r)=I1(s),I0(g)=I0(r),I0(f)=I0(g)}となる．こ

†1 以降では，枚数の都合で実体の属性を求める場合のみを議論する．しかし，この議論は識別子が複数項目からなるように変更するだけで用意に関連の属性を求める場合に拡張できる．

†2 本節においては，英小文字は要素，英大文字は集合（リスト）を表わす．また同一の文字の英小a，大A文字間にはa∈Aの関係があるとする．

†3 <K1,K2>は項目K1とK2をつなげた複合項目を表わす．

†4 リストとは重複を許す順序集合を意味する．また，断わらない限り，同値分割などによって得られるリストの順序は元のリストの順序を保存するとする．

†5 このとき，結果のリスト内の要素の順序は，リストの定義式内に表われたリスト（ここでは，F）内の順序を保存するようにする．

の多段修飾が実際にどの様なファイルを経由して行われるのかを調査することがいわゆる要求分析であり，その結果を元にこの修飾関係を満たす属性リストを求める処理をプロセスやモジュールに分割しその仕様を定めることがいわゆる設計である．

2．3　等関係式によるファイル処理の定式化

ファイル処理に対する本研究の根底となる考え方は，「更新／出力ファイルLあるいは作業領域Lにおいて，識別子I0で表される実体$I0_F$の属性$X_L.I0_F$の値は，他のファイルG0におけるこの実体$I0_{F0}$の別の属性$X0_{G0}.I0_{F0}$や，この実体$I0_{F1}$と関連Rを持つ他のファイルG1における実体$I1_{F1}$の属性$X1_{G1}.I1_{F1}.<I1',I0'>_R.I0_{F1}$や，ファイルF2が実体I2を表すが同時に実体I0との関連も表すファイルである時には実体I2の属性$X2_{F2}.<IO,I2>_{F2}$などに，関数gを適用させて求まる」ということである．

$X_L.I0_F=g(X0_{G0}.I0_{F0},X1_{G1}.I1_{F1}.<I1',I0'>_R.I0_{F1},X2_{F2}.<IO,I2>_{F2},...)$…(6)

なお，ここで同じ実体を表す識別子Iが，ファイルによって異なるかもしれないので$I1_{F1}$と$I1'_R$の様に表したが，以後は簡単化のため同一記号で表す．また，識別子が所属するファイルがFやF0のようにいろいろあるが，これは式(6)によってある左辺の$I0_F$値に対応する右辺値を計算する場合はこの値を$I0_{F0}$値に持つ実体の$X0_{G0}$などを用いることを示す．式(6)のように，左辺が単一の項目で右辺に左辺値を求める算術式や条件式を用いた等式を等関係式とよぶ．本研究ではファイル処理を，「等関係式の集合（これを等関係式仕様と呼ぶ）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」と定義する．

2．4　1パス性

等関係式仕様{$Y_L.O_F=f(X1_{F1}.I1_{F1},X2_{F2}.I2_{F2},...);...$}において，ファイルF1とF2がファイル修飾子としてある等関係式fの左辺あるいは右辺の引数を修飾していれば，ファイルF1とF2は式fで連絡していると言い，F1<f>F2と書く．F1<f>F2かつF2<g>F3ならF1<f,g>F3とし，ファイルF1とF3も式fとgで連絡していると言う．この様にして，等関係式仕様{f}によってそこに現れるファイル{F}が互いに連絡していれば，仕様{f}は閉じていると言う．

プログラムが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時，このプログラムは1パスであると言う．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3つの原因について論じる．

第1の場合は，先に説明したように等関係式仕様が多段修飾を含む場合である．この時は，式(5)のような多段修飾を持つ属性値を求めるために入れ子になったリストを内部的に保持する必要があり，このリストの長さが不定であるために作業ファイルへデータを掃き出す必要がある．この状態を回避するには，「全ての等関係式に出てくる全ての項目引数が一段修飾である」ことを要求する［一段修飾条件］．

第2の場合は，ファイル内のレコードの識別子によるソート順位の問題である．仕様中の全ての等関係式の引数が次の様に全て一段修飾としよう．

$X_L.I0_F=g(X0_{G0}.I0_{F0},X1_{G1}.I1_{F1},X2_{F2}.<IO,I2>_{F2},...)$…(7)

この時，入出力ファイルL,F,F0,F1,F2,G0,G1などが識別子I0,I1,I2などIで異なる∞にソートされている場合は，同一の実体に関する情報を異なるファイルから順に取り出して処理し出力することができない．いわゆる順序不一致の場合である．これを回避するには，「全ての等関係式に出てくる全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)」ことを要求する[順序条件].

第3の場合は，異なるファイル内のレコードが共通の識別子で修飾されているかどうかの問題である．一段修飾のEOS仕様に含まれる共通の識別子Iを持つ項目XF.IG,X'F'.IG',...のファイル修飾子F,F',...に(次節の照合ファイルとして)含まれる全てのファイル名f,g,h,...においてI以上の識別子からなるデータ構造が異なっていると，X,X',...を求めるための照合処理ができないし，これらのファイルにあるより上位の識別子J(>I)による計算の繰り返しの中にこの照合処理が入れ子状に含まれない．従って，途中結果を作業ファイルに書き出さないと処理できない．これはいわゆる構造不一致あるいは脈絡不一致といわれるものである．この状態を回避するために，「識別子Iを共有するファイルは識別子I以上≦順位で大きい識別子を全て同じ順位で共有しなければならない」ことを要求する[構造条件].

EOS/Pの後を受けモジュール設計を行うEOS/Mに与える等関係式仕様は閉じた1パス仕様である必要がある．すなわち，仕様が閉じており，一段修飾条件，順序条件，構造条件を満足する必要がある．等関係式仕様がこれらの条件を満足すれば，EOS/Mは組込まれた設計知識によりプロセスをモジュールに分割し，そのSPACE仕様を生成する．従って，プロセス設計を行うEOS/Pはジョブに対する式(6)のような多段修飾を含む等関係式仕様を受け付け，これらを一段修飾条件，順序条件，構造条件を満足する式(7)のような閉じた等関係式仕様群に分割変換するとともに，適切な中間ファイルの作成を行う必要がある．

2．5　照合ファイル

例えば図1の事例における残業を行な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(例では，残業をしたかしなかったか)を表現することがよくある．この状況は照合ファイルを用いて表現できる．照合ファイルF1&F2[I]とは，2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を照合キーにして，その値が同じレコード同志を連結した仮想的なレコードからなるファイルである．なお，1つの照合に参加するファイルの数は共通のキーを持てば3つ以上でもよい．

F1&F2[I]={f1^f2|∃f2∈F2,∃f1∈F1,I(f1)=I(f2)}†6…(8)

また照合ファイルF1&!F2[I]や!F1&F2[I]は，実体Iを表すレコードがファイルF1やF2だけにあって他方にないファイルで

†6　f1^f2はレコードf1の次にf2をつなげたレコードである．

ある．

$F1\&!F2[I]=\{f1|\forall f2\in F2,\exists f1\in F1,I(f1)\neq I(f2)\}\cdots(9)$

これらの照合ファイルは，いわゆるファイル処理における複数ファイルの照合処理を意味する．また，関係データベースにおけるジョイン演算に相当する[3]．照合ファイルは，実体Iの情報が複数のファイルに格納されている際に，これらの情報を実体毎に取り出し等関係式に従った計算を行うために必要となる．

2．6　漸化性

等関係式(6)で与えられる関数関係を解決する実際の計算について考える．等関係式内の項目は式(2)に与えられるように一般にリストである．等関係式の右辺のインスタンス間の対応は，左辺の識別子I0の各値i0に対する左辺値を，右辺の実体i0に関連する項目の値リストX={x01,x02,x03,…,x0n}に対する計算gから求めることを表している．この事を明示化するためにgの値を決める実際の計算を行なう単一の値を引数にする関数をg'(X,…)と書く．一般に，この関数は1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'が式(10)のように初期値を与えるx0と，記憶域中の中間結果g'(X')と現在レコード内の値xiからg'(X)を求めるg1に分解できる時，関数gが漸化性を持つという．

$$g'(X,\cdots)=\begin{cases} x0 & \text{Xが空リストのとき} \\ g1(xi,g'(X',\cdots),\cdots) & \text{X=[X':xi]†7のとき} \end{cases} \cdots(10)$$

X:MAX，総和:SUM，先頭値:1STなどを求めるものがあり，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わすことができる．

3．　等関係式仕様

前節で論じた式(6)や式(7)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．従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仕様を具体的に表すことにする．なお，これらの各式が関数gを適切に選べば式(6)の形式を持つことは容易に確認でる．また，どの等関係式も全て漸化性を持つことも分る．また構文的には一段修飾のEOS仕様とはファイル修飾子が全て単一ファイル修飾子の場合である．

また，識別子のファイル修飾子において照合ファイルがある場合，一般にその照合キーを明記するが，その照合キーが識別子に等しい場合は省略できる．さらに，項目のファイル修飾子がその識別子のファイル修飾子に等しい時は，それを省略できる(X_F.I_F=X.I_F).

```
<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> '}' ).
```

†7　[X':x]はリストX'にxを後につなげたリストを表わす．

```
<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>¦<一意数字定数>¦'('<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>['_'<ファイルid>][':'('#(1,3){<指標及び添
字付け>:',' } ')']'.'<識別子並び>.
<一意定数>::= 定数'.'<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>['
_'<ファイル修飾子>])[('+'¦'- ')<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>::=<データ項目名>'_'<ファイル修飾子>.
<関連識別子>::='<'#(2,-){<データ項目名> :',' } '>''_'<ファイル修飾子
>.
<ファイル修飾子>::=<単一ファイル修飾子>¦#(2,-){'('<ファイル修飾
子>')':'&'}['['<照合キー名>']'].
<単一ファイル修飾子>::=(['!']<ファイルid>)¦#(2,-){['!']<ファイルid>:'
&'}['['<照合キー名>']'].

⑤字句要素
<名前>::=(<日英字>[#(0,18){['-']<日英数字>}<日英数字>]).
<ファイルid>::=#(2,2){<英数字>}.
<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　等関係式仕様の構文規則
Fig. 2 Syntax of EOS specification.

⑮'社員No_B.@_le&b'[部No] = 社員No_b'.@_le&b'[部No];
⑯'部No_B.@_le&b'[部No] = 部No_b'.@_le&b'[部No];
⑰'エラー行1_B.@_le&b'[社員No] = FOLLOW(社員No_B.@_le&b'[社員No]^エラーメッセージ1_B.@_le&b'[社員No]);
⑱'エラー行2_B.@_le&b'[部No] = FOLLOW(部No_B.@_le&b'[部No]^社員No_B.@_le&b'[部No]^エラーメッセージ2_B.@_le&b'[部No]);
⑲'WRITE(エラー行1_B.@_le&b'[社員No]);
⑳'WRITE(エラー行2_B.@_le&b'[部No]);

図3　給与計算問題に対する等関係式仕様とEOS/Pによる変換過程
Fig. 3 EOS specification for the example salary calculation problem and its transformation by EOS/P.

4　プロセス設計の自動化

EOS/Pによるプロセス設計とは，等関係式仕様で与えられたジョブ仕様をいくつかのプロセス仕様に分割することである．この分割の基本は，まず与えられたジョブ仕様を閉じたプロセス仕様群に分割し，次に一連の中間ファイルを作成するプロセスを新たに生成することで，ジョブ仕様に含まれる多段修飾を取り除き，順序条件と構造条件を満足するようにソートプロセスや照合プロセスを追加し，生成された中間ファイルに無駄がないようにプロセスフローを最適化し，最後にジョブ仕様と等価な一段修飾の最終プロセスの仕様を生成することである．これによって生成された個々のプロセス仕様はEOS/Mによってモジュール仕様に変換できる等関係式仕様になる．以下ではこの一連の変換手続きを示す共に，その正当性を議論する．

4．1　閉じていない仕様の解決

与えられた等関係式仕様{g}が閉じていなければ，仕様{g}を仕様{g1},仕様{g2},...なるそれぞれ閉じた仕様群に分割する．次に作成された閉じた仕様{gi}毎にプロセスPiを割り当てることにする．さらに，Piの出力ファイルがPjの入力ファイルになっている時Pi→Pjなる順序を定義し，プロセス集合{Pi}をこの順序で位相ソートし，その結果の順に処理を行うようにプロセスフローを設計する．なお，ここで作成されたプロセス分割{Pi}は第一歩であり，必要なら以下の手順に従いプロセスPiをさらに部分プロセス{Pij}に分割する．

例題に対する図3(a)で与えられた仕様にはファイルE,O,R,D,M,L,B,Wを含むがこれらはいずれかの式において共有されており，仕様は明らかに閉じている．

4．2　一段修飾条件の解決

閉じた仕様{g}中のある等関係式g中に次の様な多段修飾を持つ属性

$D_F.I1_{F1}.I1_{F2}.....In\text{-}1_{Fn\text{-}1}.In_{Fn}$…(11)

があるとする．ここで，ファイルFi(2≦i≦n-1)は実体か関連を表すのでその識別子Ii_{Fi}は，そこに含まれる実体識別子をKiあるいはKi1,Ki2で表すと，Ii_{Fi}=Ki$_{Fi}$あるいはIi_{Fi}=<Ki1,Ki2>$_{Fi}$となる．なお，Fiとして複数のファイルからなる照合ファイルを用いてよいのは修飾の始点であるFnだけである．これは，Fnは実体Inの条件付けを行うために照合ファイルになることがあるが，他のファイルは修飾経路として用いられるので単一ファイルであるべきであるからである．

さて，多段修飾の属性(11)を一段修飾に直すには，式(5)に示した様に実体や関連を経由して修飾元の識別子Inから修飾先の識別子I1に至るレコードリストを逐一中間ファイルとして生成していけばよい．それではこれらの中間ファイルはどの様な演算で求められるだろうか．答えは照合処理である．すなわち，次のことが言える．

［定理1］
T=G&(R&S[I1])[I0]，U=F&T[I0]として，$X_F.I0_G.$<I1,I0>$_R.I1_S$=$X_F.I0_T$=$X_U.I0_U$である．
特に，$X_G.I0_G.$<I1,I0>$_R.I1_S$=$X_T.I0_T$である．
また，この関係は各ファイル名の前に!が付いていても同様である．
［証明］一段修飾の定義式(2)より，$X_F.I0_T$={X(f)|∃t∈T,∃f∈F,I0(f)=I0(t)}
また，照合ファイルの定義式(8)より，
T=G&(R&S[I1])[I0]={g^r^s|∃s∈S,∃r∈R,∃g∈G,I1(r)=I1(s),I0(g)=I0(r)}
これを，最初の式に代入すると，
$X_F.I0_T$={X(f)|∃s∈S,∃r∈R,∃g∈G,∃f∈F,I1(r)=I1(s),I0(g)=I0(r),I0(f)=I0(g^r^s)}となるが，この右辺は式(5)より$X_F.I0_G.$<I1,I0>$_R.I1_S$の定義そのものである．
他も同様に証明できる．
［証明終了］

ファイル修飾内での&演算については，明らかに以下の結合の法則と交換の法則が成立する．
［定理2］(A&B[I0])&C[I1]=A&(B&C[I1])[I0]　と　A&B[I0]=B&A[I0]

なお，関連を経由しない同じ実体I0を表す3つのファイルによる1つの照合A&B&C[I0]と，関連Bを経由した2つの照合(A&B[I0])&C[I1]は異なることに注意する必要がある．また交換の法則が成立するので，1つの照合においてはファイル名はアルファベット順に左から右に並べることにする．

これらの定理から，式(11)のファイルFとファイルF1の間に共通する実体識別子をK0，またファイルFiとFi+1との間に共通する実体識別子をKi(=Ki$_{Fi}$またはKi1$_{Fi}$またはKi2$_{Fi}$)とすると，
S=F1&(F2&(F3...(Fn-2&(Fn-1&Fn[Kn-1])[Kn-2])...)[K2])[K1],
T=F&S[K0]…(12)
のようなファイルを生成すれば，式(11)は
$D_F.I1_S$あるいは$D_t.I1_T$…(13)
となる．ただし，ここでtはTを構成するファイル名の1つで!付きでないもので，かつ項目Dを含むものを選ぶことにする．通常はFが識別子のファイル修飾中で!付きでなけら場Fそのものをtとして使う。また，(12)に示したように多段修飾を一段修飾に直す一連の照合においては，修飾元から修飾先への順に右から左に並べることにする．

なお，この中間ファイル生成に当たって照合される2つのファイルFiとFi+1がKiで第一順位にソートされていなければソートする必要がある．なお，与えられた等関係式を一段修飾にするには，入力ファイルの項目や定数に対しては(13)式の内後者の$D_t.I1_T$への変換を，出力ファイルや作業領域の項目に対しては前者の$D_F.I1_S$への変換を行う．なぜなら，最終のプロセス仕様に現れる入力ファイル数をできるだけ減らしたいからである．

以降において設計過程を具体的に説明するために，図1(a)に示したデータモデルにおいて，以下に示すような処理要件を持つファイル処理問題を考える．なお，EOSが扱う問題では，同一ファイル中のREDEFINEされた異なるレコード記述中では，同一項目は同じ位置に指定されているものとする．

［処理要件］

1）給与マスタに登録されている社員毎に給与明細リストの各行を出力する．この際，社員が所属する部毎に部合計も印字する．

2）給与は基本給と残業手当からなる．

3）残業手当は当月の残業代を社員ごとに集計したものである．

4）残業代は，残業記録ごとにその残業区分に対応する掛率を使って下記の式で求められる．

残業代＝単価×時間×掛率5）残業ファイルの各残業レコードに対して，同じ社員Noを持つ給与マスタレコードがなかったり，この社員が所属する部Noを持つ部レコードがなかったりする場合は，その都度”残業社員エラー”や”残業部エラー”などのエラーメッセージを出力する．

この処理要件を等関係式仕様で表現すると図3(a)のようになる．なお，ファイル修飾子は図1のデータモデルで：に続けて示されたものを用いる．また，実体や関連の識別子も図1でその前に＊印を持つ属性として表明されているとする．さらに，実体ファイルおよび関連ファイル間の同一キーを持つレコードの対応多重度をn:mの様に示した．この事例では例えば，①は残業した社員の残業手当は，この社員の給与マスタの単価と残業レコード毎の残業時間(時間$_O$.<社員No,区分,@>$_O$.社員No$_E$)とその残業の区分に依存した掛率の積をこの社員について集計したものであることを示している．②は残業しない社員の残業手当は0であることを示している．また，③では作業領域中(W)の給与は給与マスタ(E)の社員Noごとに計算され，それは社員No$_E$で表される社員の給与マスタ(E)内の基本給と社員No$_E$で表される社員の作業領域中(W)の残業手当の和であることを，⑦ではリスト(L)の部合計は部ファイル(D)の部Noごとに計算され，所属ファイル(M)を経由して部Noと関連する社員Noを持つ社員の給与マスタ(E)中の給与の和であることを，⑩ではリスト(L)の明細行は部名，社員名，給与からなることなどを示している．同様にして⑫から⑳は給与マスタに社員レコードがない残業レコードに記録された社員が所属する部が部ファイルにない残業レコード毎(@_!D&(M&O[社員No])[部No])にエラー処理を記述している．

(a)事例に対して与えられた等関係式仕様(ここでは，ファイル修飾子における照合キーや項目のファイル修飾子も全てを明記した)

①残業手当_W.社員No_E&O[社員No] = SUM(単価_E.社員No_E * 時間_O.<社員No,区分,@>_O.社員No_E * 掛率_R.区分_R.<社員No,区分>_O.社員No_E);
②残業手当_W.社員No_E&!O[社員No] = 0.社員No_E&!O[社員No];
③給与_W.社員No_E = 基本給_E.社員No_E + 残業手当_W.社員No_E;
④社員名_L.社員No_E = 社員名_E.社員No_E;
⑤給与_L.社員No_E = 給与_W.社員No_E;
⑥部名_L.部No_D = 部名_D.部No_D;
⑦部合計_L.部No_D = SUM(給与_L.社員No_E.<部No,社員No>_M.部No_D);
⑧WRITE(明細行_L.社員No_E);
⑨WRITE(合計行_L.部No_D);
⑩明細行_L.社員No_E = FOLLOW(部名_L.部No_D^社員名_L.社員No_E^給与_L.社員No_E);
⑪合計行_L.部No_D = FOLLOW(部名_L.部No_D^部合計_L.部No_D);
⑫エラーメッセージ1_B.@_!E&O[社員No] = "残業社員エラー".@_!E&O[社員No];
⑬エラーメッセージ2_B.@_!D&(M&O[社員No])[部No] = "残業部エラー".@_!D&(M&O[社員No])[部No];
⑭社員No_B.@_!E&O[社員No] = 社員No_O.@_!E&O[社員No];
⑮社員No_B.@_!D&(M&O[社員No])[部No] = 社員No_O.@_!D&(M&O[社員No])[部No];
⑯部No_B.@_!D&(M&O[社員No])[部No] = 部No_M.@_!D&(M&O[社員No])[部No];
⑰エラー行1_B.@_!E&O[社員No] = FOLLOW(社員No_B.@_!E&O[社員No]^エラーメッセージ1_B.@_!E&O[社員No]);
⑱エラー行2_B.@_!D&(M&O[社員No])[部No] = FOLLOW(部No_B.@_!D&(M&O[社員No])[部No]^社員No_B.@_!D&(M&O[社員No])[部No]^エラーメッセージ2_B.@_!D&(M&O[社員No])[部No]);
⑲WRITE(エラー行1_B.@_!E&O[社員No]);
⑳WRITE(エラー行2_B.@_!D&(M&O[社員No])[部No]);

(b) 変換された一段修飾の等関係式仕様((a)の初期仕様から変更されたもののみ示した)

①'残業手当_W.社員No_E&O[社員No] = SUM(単価_E.社員No_E * 時間_O.<社員No,区分,@>_O&E[社員No] * 掛率_R.区分_R&(O&E[社員No])[区分]);
⑦'部合計_L.部No_D = SUM(給与_L.社員No_E&(D&M[部No])[社員No]);

(c) a=R&O'[区分]なる中間ファイルを作成するプロセスのために生成された等関係式

・区分_a.社員No_R&O' = 区分_O'.社員No_R&O';
・社員No_a.社員No_R&O' = 社員No_O'.社員No_R&O';
・時間_a.社員No_R&O' = 時間_O'.社員No_R&O';
・掛率_a.社員No_R&O' = 掛率_O'.社員No_R&O';
・WRITE(レコード_a.社員No_R&O');
・レコード_a.社員No_R&O' = FOLLOW(区分_a.社員No_R&O'^社員No_a.社員No_R&O'^時間_a.社員No_R&O'^掛率_a.社員No_R&O');

(d) 順序条件と構造条件を満足する一段修飾の等関係式仕様(識別子のファイル修飾子における照合キーが明示されていない場合は，それは識別子そのものに等しい)

①"残業手当_W.社員No_e&b'[社員No] = SUM(単価_e.社員No_e * 時間_b'.<社員No,区分,@>_b'&e[社員No] * 掛率_b'.<社員No,区分>_b'&e[社員No]);
②'残業手当_W.社員No_e&!b'[社員No] = 0.社員No_e&!b'[社員No];
③'給与_W.社員No_e = 基本給_e.社員No_e + 残業手当_W.社員No_e;
④'社員名_L.社員No_e = 社員名_e.社員No_e;
⑤'給与_L.社員No_e = 給与_W.社員No_e;
⑥'部名_L.部No_e = 部名_e.部No_e;
⑦"部合計_L.部No_e = SUM(給与_L.<部No,社員No>_e);
⑧'WRITE(明細行_L.社員No_e);
⑨'WRITE(合計行_L.部No_e);
⑩'明細行_L.社員No_e = FOLLOW(部名_L.部No_e^社員名_L.社員No_e^給与_L.社員No_e);
⑪'合計行_L.部No_e = FOLLOW(部名_L.部No_e^部合計_L.部No_e);
⑫'エラーメッセージ1_B.@_!e&b'[社員No] = "残業社員エラー".@_!e&b'[社員No];
⑬'エラーメッセージ2_B.@_!e&b'[部No] = "残業部エラー".@_!e&b'[部No];
⑭'社員No_B.@_!e&b'[社員No] = 社員No_b'.@_!e&b'[社員No];

給与計算の例では，①に時間$_O$.<社員No,区分,@>$_O$.社員No$_E$と掛率$_R$.区分$_R$.<社員No,区分>$_O$.社員No$_E$が，⑦に給与$_E$.社員No$_E$.<部No,社員No>$_M$.部No$_D$の計３つの多段修飾がある．これらを一段修飾に直すために処理(12),(13),(14)を行う．この結果，初期仕様である図３(a)は図３(b)のようになる．ここで，時間$_O$,掛率$_R$については，D$_t$.I1$_T$型への変換を行うが，この際項目の修飾子は元のFをそのまま使った．また，給与$_L$はD$_F$.I1$_S$型への変換を行った。

このように多段修飾から変換された項目

D$_F$.I1$_{F1\&...\&Fn}$…(14)

は，ファイル修飾子に異なる識別子による複数の照合を含んでいるので正確には一段修飾ではない．このことは⑬⑮⑯⑱⑳における識別子@$_{!D\&(M\&O[社員No])[部No]}$についても同様である．これらを一段修飾にするには，前処理プロセスにおいて一連の中間ファイルtn-1=Fn-1&Fn,...,ti=Fi&ti+1,...を作成し，単一ファイル修飾子t1=F1&t2をファイル修飾に用いる必要がある．この際，必要な前処理プロセスの数をできるだけ減らすように設計することが重要である．

結合の法則からD$_F$.I1$_{F1\&...\&Fi\&Fi+1\&...\&Fn}$=D$_F$.I1$_{(F1\&...\&Fi)\&(Fi+1\&...\&Fn)}$である．従って，中間ファイルとしてs1=F1&...&Fiとs2=Fi+1&...&Fnを用いれば，式(14)はさらに

D$_F$.I1$_{s1\&s2[Ki]}$…(15)

となる．

この式(15)においては１つの識別子による照合しか含まれない，しかもこの照合は最終プロセスにおいて行うことができる．従って，t=F1&...&Fi&Fi+1&...&Fnを中間ファイルに使うよりも，s1とs2を使った方が前処理プロセスの数を減らすことができる．残る問題はどの様な基準の元に中間ファイルへの分割を行うかである．この基準として重要なのは，最終プロセスに入力するファイル（最終ファイルという）の数をできるだけ少なくすることである．具体的には，最終ファイルを見つける手順は以下の様になる．

1)見かけの一段修飾のEOS仕様における識別子のファイル修飾子群modif内にF&G&Hと!F&G&!Hのように複数のファイル間の相異なる&結合が最も多く出現するファイルの組合わせを見付け，その構成ファイルを最終ファイルの根rootとして登録する．なお，どの根も含まないファイル結合がmodif内に存在するなら，そのファイル結合の構成ファイルの１つを根としてrootに加える．次に各根rから生成される最終ファイル要素集合term(r)の初期値をrとする(term(r)={r})．例：modif={E&O,E&!O,O&E,R&O&E,E,E&M&D,!E&O,!D&M&O}(照合キーは省略した)には，EとO，DとMの照合が出現しているが，前者の方が多く用いられているので，root={E,O}とする．また，modif内のどのファイル結合もEかOを含んでいるので，根としては充分である．

2)rootに含まれない入力ファイルFについて，ある根rのterm(r)内のファイルあるいはファイル結合tに対してF&tやt&Fがmodif内のあるファイル結合に部分結合として含まれるなら，term(r)にFとF&tあるいはt&Fを加える．ただし，modif内にF&tとして出現している時はこの操作は行わない．このようにしてterm(r)は成長する．この結果，最終プロセスに入力する最終ファイルtermsを，各term(r)内の全ての単一ファイルをtermに加えられた順(根に繋げられた順)に右から並べて&結合したものとする(terms={&(term(r))|r∈root})．

例：Rに対してO∈term(O)かつR&O⊂R&O&E∈modifより，term(O)={O,R,R&O}．Dはすぐ結合できないので後回し．Mに対してはE∈term(E)かつE&M⊂E&M&D∈modifより，term(E)={E,M,E&M}，またO∈term(O)かつM&O⊂!D&M&O∈modifより，term(O)={O,R,R&O,M,M&O}．Dはこの段階で，E&M∈term(E)かつE&M&D⊂E&M&D∈modifより，term(E)={E,M,E&M,D,E&M&D}．最後に&(term(E))=D&M&E，&(term(O))=M&R&Oであるので，terms={M&R&O,D&M&E}となる．

この様な最終ファイルterms={M&(R&O),D&(M&E)}は，実際には図１に示すように，O'=sort(O,<区分,社員No>);a=R&O'[区分];a'=sort(a,<社員No,区分>);m'=sort(M,<社員No,部No>);b=m'&a'[社員No];c=m'&E[社員No];c'=sort(c,<部No,社員No>);e=D&c'[部No];なる一連の中間ファイルを経由して行われる．

最後に，項目および識別子のファイル修飾を最終ファイルを使ったものに置き換える．すなわち，modif内の各ファイル修飾f=...&g&...内の部分結合gで，交換や結合の法則を考慮してどれかのterm(r)に含まれているものの中で最長のもの見つけ，fにおいてgを最終ファイル&(term(r))で置き換える．ただし，この時項目のファイル修飾子も最終ファイルで置き換えるが，このファイル修飾は基本的には単一の最終ファイル名である必要がある．このようなファイル名としてのgが複数ある時は，その項目の識別子のファイル修飾子において!なしで用いられているものを使う．

例：E,D,E&M&D→e=D&M&E，およびM,O,R&O,M&O→b=M&R&Oなる置き換えを行うことで，E&O[社員No]→e&b[社員No],E&!O[社員No]→e&!b[社員No],O&E[社員No]→b&c[社員No],R&(O&E[社員No])[区分]=(R&O[区分])&E[社員No]→b&e[社員No],E→e,D→e,O→b,E&M&D→e,!E&O[社員No]→!e&b[社員No],!D&(M&O[社員No])[部No]→!e&b[部No]となる．なお，ここで⑯の部No_MのMとしてはeでもbでもよいのであるが，識別子は!e&bとなるので，!の付いていないbを用いる．また，同じ照合ファイル!e&bが最後に２つあるがこれらは照合キーが異なっていることに注意しよう．この違いはそれらが修飾する識別子として式⑭'や⑮'に反映されている．

ただし，ここで１つ問題がある．それはg→t=&(term(r))なる置き換えがレコードの過不足に関して正しいかどうかである．確かにtは部分列としてgを含むのでgファイルのレコード構成項目はtファイルのレコードにも含まれる．しかし，t=...&h&...&g&...を照合処理によって作成する段階で，tにおいてgより左に位置するhに対して，gに対応するがhに対応しないtレコードは落ちこぼれるし，またh:gがn:1対応ならgの１レコードに対してn個のtレコードが生成される．この不都合を解決するために，まず最終ファイルを作成する照合処理においては&結合ではなく以下の様な%照合を使う．

F1%F2[I]=F1&F2[I]+!F1&F2[I]…(16)

なお，式(16)の%照合の結果ファイルの右辺第２項から来るレコードにおいては，F1レコードに含まれる項目に対応する項目の値は未定義値(-)とする．こうすれば明らかにF2⊂F1%F2なので，F2→F1%F2の置き換えにおいてF2のレコードが落ちこぼれることはない．さらに，g→tの置き換えに際してgの１レコードに対しn個のtレコードが存在する時は，g→t'=1st(sort(t,J),J)の置き換えとする．ここで，1stは識別子Jが等しいレコードグループの先頭レコードを取り出す関数であ

り，またJはファイル結合g=g1%...%gnの最右端のファイルgnの識別子である．こうすることによって，t'の多重度はgと等しくなる．

例：E→e=D%M%Eなる置き換えにおいて，Eより右側のM,Dとの結合においても対応はn:1ではなく1:1か1:nなので，1st演算を行う必要はない．しかし，D,E&M&D→e=D%M%Eの置き換えにおいてはDの左側にM,Dが結合しているので，多重度が増える．しかしこのファイル修飾子を持つ式は，⑥⑬⑮が転記式，⑪⑱がfollow式，⑨⑫が右辺のない式であるので多重度が増えても計算結果に不都合はない．またO,R&O,M&O→M%R%O=bなる置き換えにおいて，Oより右側のM,Rとの結合においても同様である．

なお，多段修飾の解決のために最終ファイルを作成する前処理プロセスにおいては%による照合を行うが，これらを用いたEOS仕様においては&を用いる．

4．3　順序条件の解決

閉じた一段修飾の等関係式仕様に現れる相異なるファイル修飾子FとGを共通して持つ識別子が複数あったとしよう．両ファイルでこれらの識別子によるソート順位が異なっていれば，これらの一方をソートして他方におけるソート順位に合わせる必要がある．重要なのはどのファイルにおける順位に合わせるかである．基準としては，まず出力ファイルにおける順位に合わせる，共に入力ファイルなら中間ファイルを外部ファイルに合わせる，共に外部や中間ファイルなら識別子の多い方に合わせる．例えば，Fがこれらの条件のどれかを満たす基準となるファイルであり，そこでのソート順位がI0,I1,I2,...Inなら，ファイルGに対して，

G'=sort(G,<I0,I1,...,In>);…(17)

なるsort処理を行う．

給与計算に対する一段修飾の等関係式仕様に現れるファイルは，L,B,b,eである．ここで，Wは作業領域でファイルではないので対象としない．これらのファイルに現れる識別子は"部No","社員No","区分"の３つである．各ファイルはこれらの識別子で，L,Bでは"部No","社員No"の順，bでは"区分","社員No","部No"の順，edでは"部No","社員No"の順にソートされている．基準となるファイルとしては，B,Lが出力であるのでここに現れる識別子に対しては"部No","社員No"の順となる．従って，これらに合わせるためにb'=sort(b,<部No,社員No,区分>);なる処理を行う．

4．4　構造条件の解決

閉じた一段修飾の順序条件を満足する等関係式仕様内の，相異なるファイルF0とF1に対して，これらをファイル修飾子に持つ同一実体の識別子I_{F0}とI_{F1}が構造条件を満足していないとしよう．すなわちこれらのファイル間で識別子Iの現れる≦順位が異なるとしよう．これは，F1とF2の間で識別子Iのソート順位が異なるからである．例えば，F1がI11,I12,...,I1m,I,I1(m+1),...で，F2がI21,I22,...,I2n,I,I2(n+1),...でそれぞれこの順にソートされているとしよう．順序条件を維持するためにこのソート順位を換えることはできない．従ってできるのはファイルF1,F2に項目を追加し，それぞれG1,G2なる中間ファイルに拡張し，両ファイルを同じ識別子で同じ順位でソートされているようにすることである．Fi内でIとIijが共に項目として存在する以上，それらを関連づける関連ファイルが外部ファイルとして存在しているはずで，このファイルをFijとする．するとこの処理は次の様になる．

G10=sort(F1,<I>);G11=G10&F21[I];...;G1n=G1(n-1)&F2n[I];…(18)

さらに，I11,I12,...,I1m,I21,I22,...,I2nを≦でトポロジカルソートし，結果をI1,I2,...,I(m+n)とすれば，

G1=sort(G1n,<I1,I2,...,I(m+n),I>);…(19)

なる一連の処理を行う．同様にして，G2も作成する．これは一言でいえば，F1とF2がIを含む共通の識別子対<I1,I2,...,I(m+n),I>を持つように拡張することであり，これによって両ファイルが照合可能になる．なお，Fi→Giなる置き換えは照合による大きい識別子の追加なのでレコード件数に変化はなく置き換えても題意に変化はないことが分る．

給与計算に対して先の順序条件を解決するために作成した中間ファイルb'でその元となったbを入れ替えると，ファイルはL,B,b',eになる．これらは全て，"部No","社員No","区分"の部分順序であり，上部を共有しているので構造条件を満足している．

以上の一連の変換によって，閉じた仕様{gi}に対するプロセスPiは，情報を適切に含んだ一連の中間ファイルを作成する前処理プロセスPijと，これらの最終的に得られた中間ファイルを用いて仕様に与えられた計算を行う最終プロセスP'iからなる一連の処理に分解される．例題はそれ自身閉じているので当初１つのプロセスとして始まるが，１パス性を持つプログラムを生成するために図１(b)に示すような照合プロセスやソートプロセスを含む前処理プロセス群に分割される．簡単な問題の割には中間ファイルが多数必要になったように思えるが，これはEOS/Pの機能を説明するためにわざと事例のデータモデルにおいて情報を分離して複数のファイルに格納したためである．実務で使われるデータモデルではもう少し冗長な識別子をファイルに持たせているので，構造条件等の解決のために中間ファイルを作る必要がない場合が多い．

EOS/Pはこれらの処理を自動的に行い，結果として中間ファイルO',a,a',b,b',c,c',e,m'を作成するプロセスに対する等関係式仕様を生成する．例えば，a=R&O'[区分]なる処理に対しては，図３(c)のような等関係式を生成する．なお，このような中間ファイルを作成する時，中間ファイルの項目が照合用の両ファイルに共通する場合は右側の照合ファイル(例ではO')の項目を使う．さらにEOS/Pは，生成された最終中間ファイルb'とeを用いて，図３(d)に示すような順序条件と構造条件を満足した一段修飾の等関係式仕様を生成する．

5．　おわりに

本研究によって，等式集合として非手続き的に与えられた仕様から，データの修飾関係とファイルにおける格納構造をもとに，この仕様を満足するファイル処理を一連のプロセスに分割し，各プロセスと中間ファイルの正確な仕様を決定する理論を構築できた．ここで，各プロセスは外部ファイルにデータを書出さずに処理できる１パス性を持ち，生成された仕様ではいわゆる順序不一致や構造不一致は解決されており，不必要な中間ファイルは作成しないという意味で最適な設計になっている．また，この理論に基づいた自動設計システムEOS/Pを作成し例題に適用した結果，熟練者と同程度の良好な設計結果が得られたことからその有効性を実証できた．

なお，EOS/Pによって生成された等関係式仕様からプログラムへの変換についてはEOS/M[6,7]とSPACE[5]が行う．具体的には，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．

Mod7　　CondExp = MC　　Module->FileModifier E&O

C1	MC (給与マスタE;<部No>)	Y	Y	Y	N	N	N	E	E	E
C3	MC (残業ファイルO;<部No>)	Y	N	E	Y	N	E	Y	N	E
A1	exec　　Mod8.	X	X	X						
A2	エラーメッセージ2 OF エラーリスト := "残業部エラー".				X		X			
A3	部No OF エラーリスト = 部No OF 残業ファイルO.				X		X			
A4	社員No OF エラーリスト = 社員No OF 残業ファイルO.				X		X			
A5	XWRITE エラー行2 OF エラーリスト.				X		X			
A6	do_again.				X	X	X	X	X	

Mod8　　CondExp = MC　　Module->FileModifier E&O

P1	IF DT07--MC--<部No> NOT = 給与マスタE--<部No>--C THEN EXIT.									
P2	IF DT07--MC--<部No> NOT = 残業ファイルO--<部No>--C THEN EXIT.									
C1	MC (給与マスタE;<部No,社員No>)	Y	Y	Y	N	N	N	E	E	E
C3	MC (残業ファイルO;<部No,社員No>)	Y	N	E	Y	N	E	Y	N	E
A1	exec　　Mod0.	X								
A2	残業手当 OF 作業領域 := 0.		X	X						
A3	給与 OF 作業領域 := 基本給 OF 給与マスタE + 残業手当 OF 作業領域.	X	X	X						
A4	給与 OF 給与明細リスト := 給与 OF 作業領域.	X	X	X						
A5	社員名 OF 給与明細リスト := 社員名 OF 給与マスタE.	X	X	X						
A6	do　　Mod3.	X	X	X						
A7	XWRITE 明細行 OF 給与明細リスト.	X	X	X						
A8	エラーメッセージ2 OF エラーリスト := "残業社員エラー".				X			X		
A9	社員No OF エラーリスト = 社員No OF 残業ファイルO.				X			X		
A10	XWRITE エラー行2 OF エラーリスト.				X			X		
A11	do_again.	X	X	X	X	X	X	X	X	

Mod3　　CondExp = LC　　Module->FileModifier E

C1	LC (給与マスタE;<部No>)	F	I	L	U	E
A1	部名 OF 給与明細リスト := 部名 OF 給与マスタE.	X			X	
A2	部合計 OF 給与明細リスト := 0.	X			X	
A3	部合計 OF 給与明細リスト := 部合計 OF 給与明細リスト + 給与 OF 給与明細リスト.	X	X	X	X	
A4	XWRITE 合計行 OF 給与明細リスト.			X	X	

Mod0　　CondExp = CB　　Module->FileModifier E&O

C1	CB (残業ファイルO;<部No,社員No>)	H	B	T	E
A1	残業手当 OF 作業領域 := 0.	X			
A2	残業手当 OF 作業領域 := 残業手当 OF 作業領域 + 単価 OF 給与マスタE * 時間 OF 残業ファイルO * 掛率 OF 残業ファイルO.		X		
A3	do_again.	X	X		

図4　給与計算問題に対してEOSが生成したSPACE用のモジュール仕様群

Fig. 4 SPACE module specifications generated by EOS for the sample salary calculation problem.

例えば，図3(d)に与えられた仕様からは4モジュールからなる図4のようなSPACE仕様が生成される．ここでは，モジュール7においてファイルe,b'がMC条件式を用いて部Noレベルで照合され，MC式の値がY,N,Eを取る各組合わせ下においてモジュール6を呼出したり，エラー処理をする．モジュール6は呼出されると同じくファイルeとファイルb'間が社員Noレベルで照合され，MC式の値がY,N,Eを取る各組合わせ下において，すなわち社員が残業したかしなかったかにおいて，モジュール3やモジュール0を呼出したり，明細行を書き出したりする．また，モジュール3はLC条件式を用いて繰り返しの各状態下において部合計の初期化や加算を行ったり合計行を出力し，同様にモジュール0はCB条件式を用いて残業手当の集計計算をしている．

今後の課題としては，本研究で解決できなかった1パス性の条件であるバックトラック問題を解決すること，EOS/Pが行ったプロセス分割の結果をSPACEのプロセスフロー図に変換すること，生成された中間ファイルのレコードレイアウトをSPACEのコピーライブラリやファイルテーブルやデータ構造テーブルに登録することなどの自動化が残っている．

謝辞

本システムの前提になったEOS/MおよびEOS/P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)，浅見伸美(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. Software Eng.,vol.SE-11,No.11,pp.1321-1336(1985).
(2) 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).
(3) E.F.Codd:"A Relational Model of Data for Large Shared Data Banks",Commun. ACM,Vol 13,No.6,pp377-387(1970).
(4) 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",情処学論,Vol.27,No.4,pp.417-424(1986).
(5) 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.J71-D,No.7,pp.2555-2562(1988).
(6) 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.J72-D-1,No.4,pp.262-271(1989).
(7) 原田実,中村義幸:"プログラムの構造と論理の自動設計システムEOS/M",情処学論,Vol.34,No.9,pp.2006-2018(1993).
(8) Jackson M.A.:"Principles of Program Design",Academic Press(1975).
(9) 黒木宏明,磯田定宏:"統合化CASEシステムSoftDAの機能とその評価",情処学ソフトウェア工学研報92-SE-83,pp17-24(1992).
(10) 日経BP社:"PRIDE-ASDM",日経データプロ・ソフト,NS2-101-013-021(1986).