非手続き型言語と入出力データの構造不一致†

橋本 正明††

プログラムの入出力データのあいだの構造不一致が非手続き型言語へ及ぼす影響について議論する．さらに，筆者がすでに提案したプログラム仕様記述法 PSDM における構造不一致の取扱いについて報告する．さて，非手続き型言語には利用者が構造不一致を意識しなければならないものがある．しかし，不一致はプログラム構造の決定要因であり，プログラム仕様の決定要因ではないので，利用者が不一致を意識するのは望ましくない．そこで，利用者が不一致を意識しなくてよい言語もある．ところが，不一致を意識することを不要としたのに起因して，言語の理解性や記述性の問題が指摘されている．また，このような言語からプログラムを生成するには，プログラム・ジェネレータが不一致を検出して解決しなければならない．ところで，PSDM で規定された言語でも利用者は不一致を意識しなくてよい．しかも，データが表している情報に着目した仕様も記述するので，言語の理解性や記述性の問題が緩和されている．この言語からプログラムを生成するには，プログラム仕様に基づいて作成された有向グラフを解析して，構造不一致を検出し解決する．この方法は，まだ構造不一致の検出対象が脈絡不一致に限定され，しかも検出精度に向上の余地は残っているが，ジェネレータを作成した実験の結果，実用的な性能を持つプログラムを生成することについて見通しを得ることができた．

1. まえがき

プログラムの入出力データのあいだに**構造不一致**[1]が存在する場合，その不一致を解決するため，たとえば配列などで定義された主記憶上のテーブルへ，処理の途中に中間データを一時蓄積するようにプログラムを作成する．中間データが多ければ主記憶のかわりにファイルを用いることもある．このように構造不一致はプログラム構造の決定要因になっている．ところで，**非手続き型言語**からプログラムを生成するには，構造不一致を決定要因とするプログラム構造を自動的に決めなければならない．このため，構造不一致は非手続き型言語へ種々の影響を及ぼしている．

本論文ではその影響について考察し，1) 言語の利用者が構造不一致を意識することの要否，2) プログラムの記述法，3) プログラムの生成方法，へ影響が及んでいることを明らかにする．

さて，非手続き型言語 Basic Lucid[2] はプログラム変数としてスカラ変数しか持たず，テーブルを定義できないので，利用者が構造不一致のないことを確認しなければ言語を利用できない．なお，Lucid へ配列変数を導入できる[3]が，配列変数とスカラ変数を使い分けるには利用者が不一致の箇所を検出しておかなければならない．しかし，構造不一致はプログラム構造の決定要因であり，プログラム仕様の決定要因ではないので，利用者が構造不一致を意識するのは望ましくない．

一方，非手続き型言語 MODEL[4] は配列変数しか持たず，入出力データや中間データをすべて配列で記述するので，利用者は構造不一致を意識せずに言語を利用できる．そして，ジェネレータが不一致を検出し解決してプログラム構造を決め，プログラムを生成する．しかし，データへ常に配列の添字が付いて添字の取扱いが複雑になるので，言語の理解性や記述性の問題が指摘されている[4]．

筆者は理解性や記述性がよい非手続き型言語を開発するため，すでに**計算指向 EAR** (Entity Attribute Relationship) **モデル**[5]による情報構造記述を導入したプログラム仕様記述法 **PSDM** (Program Specification Description Method)[6),7] を提案した．そこで，PSDM における構造不一致の取扱いについても本論文で報告する．

第2章には構造不一致が非手続き型言語へ及ぼす影響について述べる．第3章では PSDM を概説し，PSDM で規定された言語の利用者が構造不一致を意識しなくてよく，しかも情報構造の記述によって理解性や記述性の問題が緩和されることを述べる．ところが，情報構造記述を導入したため，構造不一致の新たな検出方法が必要となった．このため，構造不一致のうち，まず脈絡不一致[1]を対象にして検出方法を考案した．まだ，検出精度に向上の余地は残っているが，この検出方法を中心にして第4章にプログラム生成方法を説明する．ジェネレータを作成した実験について

† Nonprocedural Languages and Structure Clash among Input-Output Data by MASAAKI HASHIMOTO (ATR Communication Systems Research Laboratories).

†† (株)ATR 通信システム研究所

も述べる．

2. 構造不一致の影響

2.1 構造不一致

構造不一致の説明のため，以下に述べるプログラムを例として用いる．プログラムの入力には売上げファイルと商品ファイルがあるものとする．前者は1回の売上げにおける売上げ番号と品名，売上げ数が書き込まれた売上げレコードを多数格納し，後者は一つの商品の品名と単価が書き込まれた商品レコードを多数格納している．プログラムは両ファイルを入力し，売上げごとの売上げ額を計算した報告書を出力する．なお，両ファイルはソートされていない順呼び出しファイルとする．また，商品ファイルのデータはすべて主記憶上のテーブルへ蓄積できる程度の量とする．

ここで，売上げごとの売上げ額を計算するには，同じ品名が書き込まれたレコードを売上げファイルと商品ファイルから入力し，そのなかの売上げ数と単価について積をとらなければならない．しかし，両ファイルはソートされていないので，同じ品名が書き込まれたレコードを両ファイルから同期して入力することはできない．したがって，このプログラムの入出力データのあいだに構造不一致が存在する．

このプログラムはCOBOLなどの手続き型言語を用いて以下のように作成できる．まず，主記憶上に商品テーブルを配列で定義し，商品ファイルのデータをすべて蓄積する．次に，売上げファイルから入力したレコードに書き込まれている品名で商品テーブルをサーチし，その品名の単価を得て売上げ額を計算する．そのあと，売上げ番号と売上げ額を出力する．この処理を売上げファイルのレコードごとに繰り返す．

以上のように構造不一致は主記憶上のテーブルを用いて解決できる．テーブルのかわりに乱呼び出しファイルを用いてもよい．入力ファイルをソートするのも解決方法の一つである．

2.2 Lucid

単一代入の概念[8]を用いた Basic Lucid は配列変数を持たず，スカラ変数しか持たない．しかし，同じ代入文群を繰り返して実行でき，繰り返しの世代ごとに変数へデータを再代入できる．すなわち，繰り返しの一世代のみへ単一代入の概念を適用している．

このため，順呼び出しファイルから繰り返して入力される多くのデータを処理できる．しかし，変数へ代入された最新世代のデータは参照できても，旧世代のデータは消えてしまうので参照できない．旧世代のデータも主記憶上へ蓄積するため，すべてのデータへスカラ変数を割り当ててテーブルを定義するのは現実的でない．したがって，前節のような構造不一致が存在するプログラムは記述できず，利用者は不一致がないことを確認しておかなければ言語を利用できない．

なお，Lucid へ配列変数を導入した場合，それで主記憶上のテーブルを定義すれば，構造不一致が存在するプログラムも記述できる．しかし，不一致をスカラ変数で解決できないのにかわりはないので，配列変数とスカラ変数を使い分けるため，利用者は構造不一致の箇所を検出しておかなければならない．

2.3 MODEL

MODEL は配列変数しか持たないので，入出力データや中間データをすべて配列で記述しなければならず，しかも単一代入の概念を適用しているので，配列要素には各々一度しかデータを代入できない．このため，配列名と添字でデータがすべて識別される．

さて，2.1 節で述べたプログラムは MODEL で以下のように記述できる．入力データ用には売上げレコードが繰り返す配列 “売上げ” と，商品レコードが繰り返す配列 “商品” を記述する．出力データ用には売上げごとに売上げ番号と売上げ額が繰り返す配列 “報告書” を記述する．添字値の範囲として，配列 “商品” には商品レコード数の上限値を指定し，その他は不定とする．売上げ数と単価から売上げ額を得るための計算式は，配列名と添字変数，および IF-THEN を用いて

```
HOUKOKUSHO.URIAGEGAKU(U)=
  IF URIAGE.HINMEI(U)=
          SHOHIN.HINMEI(S)
  THEN URIAGE.KOSU(U)*
          SHOHIN.TANKA(S);
```

と記述する．ここで S と U は添字変数である．

プログラムの生成では，配列のあいだをデータが流れる順序にしたがって，計算式やファイル入出力の実行順序が決まる．その計算式やファイル入出力の各々に対しては，配列要素ごとに計算やレコード入出力を繰り返すための制御ループが一つずつ生成される．

そこで，添字値が同じ範囲を持ち，しかも順に実行される計算式やファイル入出力は同じループに入れられる．この計算式やファイル入出力のあいだでは，ループが繰り返すつど制御が受け渡される．この時のデータの受渡しを見ると，そのループのなかでのみ用

いられる配列については，一つの配列要素のデータしか受け渡されない．すなわち，配列要素ごとに見たデータの書き込みと参照が同期するので，その配列には構造不一致が存在しない．そのような配列は自動的に検出され，一要素分の主記憶しか割り付けられない．

その他の配列については，配列要素ごとに見たデータの書き込みと参照が同期しないので，構造不一致が存在する．このため，全要素分の主記憶を割り付けたテーブルが生成され，不一致が解決される．前述のプログラムでは配列“商品”に対してテーブルが生成され，不一致が解決される．ところで，2.1 節ではプログラムの入出力データのあいだの関係として構造不一致の存否が決まった．一方，MODEL では前述のように配列に局所化して構造不一致の存否が決まる．

MODEL では上記の計算式のようにデータ入力の同期性と関係なく，配列名と添字で任意の入力データを直接参照できる．このため，利用者は構造不一致を意識せずに言語を利用できる．一方，前述の主記憶割り付け方法からわかるように，ジェネレータが不一致を検出し解決してプログラムを生成する．しかし，データへ常に添字が付いて添字の取扱いが複雑になるので，言語の理解性や記述性の問題が指摘されている．したがって，利用者が構造不一致を意識しなくてよく，しかも理解性や記述性のよい非手続き型言語が望まれる．

3. PSDM

PSDM ではプログラム仕様を機能仕様[5)~7)]と実現仕様に分けて記述する．前者はプログラムの入出力データのあいだの論理的な関係を定め，後者は，前者から実行効率がよいプログラムを生成するための付帯条件を定めるものである．ところで，PSDM に基づく非手続き型言語 PSDL-2 を実験用に規定したので，2.1 節で述べたプログラムの例を PSDL-2 で記述した図 1 のプログラム仕様を用いて，以下に PSDM を概

```
PROGRAM URIAGEGAKU;
INFORMATION-LAYER;
  ENTITY-TYPE SHOHIN;
    DESCRIPTION HINMEI[ID,HINMEI,STR,,],TANKA[,YEN,NUM,,];
      ENT-NUMBER 100;
  RELSHIP-TYPE URIMONO;
    COLLECTION /SHOHIN,/URIAGE;
        CARDINALITY M,1;
      VALUE-DEPN /URIAGE/KINGAKU <- MULT[/URIAGE/KOSU,/SHOHIN/TANKA];
  ENTITY-TYPE URIAGE;
    DESCRIPTION BANGO[ID,BAN,NUM,,],KOSU[,KO,NUM,,],KINGAKU[,YEN,NUM,,];
DATA-LAYER;
  ITERATION-TYPE[ROOT] SHOHIN-DATA[E] ::= SHOHIN-R;
      ITERATE-CONS[DATA] E = 9999;
    SEQUENCE-TYPE[REC] SHOHIN-R ::= E-HINMEI,E-TANKA;
      ELEMENT-TYPE E-HINMEI PIC X(10);
      ELEMENT-TYPE E-TANKA PIC 9(6);
    ENT-CONS SHOHIN/HINMEI <-> E-HINMEI;
      VALUE-CONS SHOHIN/TANKA[HINMEI <-> E-HINMEI] <-> E-TANKA;
  ITERATION-TYPE[ROOT] URIAGE-DATA[U] ::= URIAGE-R;
      ITERATE-CONS[DATA] U = 9999;
    SEQUENCE-TYPE[REC] URIAGE-R ::= U-BANGO,U-MONO,U-SU;
      ELEMENT-TYPE U-BANGO PIC 9(4);
      ELEMENT-TYPE U-MONO PIC X(10);
      ELEMENT-TYPE U-SU PIC 9(4);
    REL-ENT-CONS URIMONO//URIAGE/BANGO <-> U-BANGO
                 WHERE /SHOHIN/HINMEI <-> U-MONO;
      VALUE-CONS URIAGE/KOSU[BANGO <-> U-BANGO] <-> U-SU;
  ITERATION-TYPE[ROOT] URIGAKU-DATA[S] ::= URIGAKU-R;
      ITERATE-CONS[ENTITY] URIAGE;
    SEQUENCE-TYPE[REC] URIGAKU-R ::= S-BANGO,S-KINGAKU;
      ELEMENT-TYPE S-BANGO PIC 9(4);
      ELEMENT-TYPE S-KINGAKU PIC ¥(10);
    ENT-CONS URIAGE/BANGO <-> S-BANGO;
      VALUE-CONS URIAGE/KINGAKU[BANGO <-> S-BANGO] <-> S-KINGAKU;
ACCESS-LAYER;
  DATASET-TYPE[FILE]  SHOHIN-F,INPUT,16,SHOHIN-DATA;
  DATASET-TYPE[FILE]  URIAGE-F,INPUT,18,URIAGE-DATA;
  DATASET-TYPE[FILE]  URIGAKUF,OUTPUT,14,URIGAKU-DATA;
END-PROGRAM;
```

図 1 プログラム仕様
Fig. 1 A program specification.

説する．それから，構造不一致が PSDM へ及ぼす影響について述べる．

3.1 機能仕様

機能仕様は以下の情報層とデータ層，アクセス層に分けて記述する．

（1） 情 報 層

プログラムの入出力データが表している対象世界の情報について，その枠組みを定めるための情報構造を計算指向 EAR モデルに基づいて記述する．図1では INFORMATION-LAYER 文に続けて記述している．

対象世界の情報は事物，事物の性質，および事物相互の対応づけとしてとらえ，各々を主体，属性および関連と呼ぶ．主体または関連の集合は各々主体型または関連型で定める．この型は主体または関連の種類を示すものである．図1の ENTITY-TYPE 文は主体型“商品”と“売上げ”を定めている．RELSHIP-TYPE 文は，COLLECTION 文に記述した二つの主体型を対応づける関連型“売物”を定めている．

この売物のように異なる主体型を対応づけた関連型もあれば，同じ主体型を相互に対応づける関連型もある．たとえば，親子関係は同じ主体型“人”を相互に対応づけた関連型である．ここで，親子関係にある二つの主体の一方は親の役割を果たし，他方は子の役割を果たす．そこで，ある役割を果たす主体の集合を“役割／主体型”の形で記述する．異なる主体型を対応づけた関連型については役割を省略してよい．

図1の DESCRIPTION 文は主体型ごとに属性を定める．たとえば，商品については品名と単価を定めている．この品名や単価の属性値で個々の主体“商品”の性質を表す．主体型のなかで一意に主体を識別するための属性を主キーと呼ぶ．DESCRIPTION 文の［ ］の最初に ID をつけた属性が主キーである．たとえば，商品の主キーは品名である．［ ］の2番目の HINMEI や YEN は属性値の定義域である．3番目の STR または NUM は定義域が文字列または数値からなることを定める．ある主体型に属する主体の属性は“主体型／属性”の形で記述し，ある主体型のなかで，ある役割を果たす主体の属性は“役割／主体型／属性”の形で記述する．

さて，関連で対応づけられたいくつかの主体について，その属性の値から他の非主キー属性の値を得るための計算方法は属性値従属性制約で定める．図1の VALUE-DEPN 文は，関連“売物”で対応づけられた主体“商品”と“売上げ”について，売上げの個数と商品の単価の積として売上げの金額の値を得ることを定めている．この文の MULT は積をとるための関数である．

また，いくつかの主体について，その主体を対応づける関連を得るための計算方法は関連存在従属性制約で定める．さらに，ある主体に基づいて，他に存在する主体を得るための計算方法は主体存在従属性制約で定める．この制約は各々 REL-DEPN 文と ENT-DEPN 文で記述するが，図1にその例は含まれていない．

（2） デ ー タ 層

COBOL のデータ部と類似な考え方でプログラムの入出力データの形式を記述する．図1では DATA-LAYER 文に続けて記述している．

まず，基本項目は基本項目型で定め，そのデータ形式を COBOL の PICTURE 句で定める．COBOL の OCCURS 句で記述する集団項目は繰り返し集団項目型で定める．この場合，繰り返しデータを識別するための指標も定める．COBOL の REDEFINES 句で記述する集団項目は選択集団項目型で定める．OCCURS 句や REDEFINES 句は用いずに同じレベル番号で続けて記述する集団項目は連接集団項目型で定める．図1では各々 ELEMENT-TYPE 文，ITERATION-TYPE 文，SELECTION-TYPE 文，SEQUENCE-TYPE 文で記述している．

以上の型をまとめてデータ型と呼ぶ．データ型は階層的に定めるので，一つのファイルのデータ形式は，データ型からできた一つの木構造で表される．そこで，図1の［ROOT］は木構造の根となるデータ型を示す．また，［REC］は入出力レコードとなるデータ型を示す．

繰り返し集団項目型についてはデータの繰り返し数を決めるための繰り返し制約も定める．たとえば，図1の ITERATE-CONS［DATA］文は指標の値が 99999 になると繰り返しが終わるように定めている．それ以前にファイルの終了が検出されても繰り返しは終わる．また，ITERATE-CONS［ENTITY］文は主体型“売上”に属する主体の数だけデータが繰り返すように定めている．選択集団項目型についてもデータ型の選択方法を選択制約で定める．この制約は SELECT-CONS 文で記述するが，図1にその例は含まれていない．

さて，プログラムの入出力データは(1)項で述べた

ように，主体，その属性値および関連を表している．このため，基本項目型や指標と，情報層で定めた型を以下のように対応づけて，データ層と情報層の関係を定める．1) ある主体型に属する主体を表している基本項目型や指標は，その主体型の主キー属性へ対応づける．2) ある関連型に属する関連と，その関連で対応づけられたいくつかの主体を表している基本項目型や指標は，その主体の主キー属性へ対応づける．3) ある主体型に属する主体の属性の値を表している基本項目型や指標は，その属性に対応づける．

上記の 1)，2)，3) は各々，主体に関する情報制約，主体と関連に関する情報制約，属性値に関する情報制約と呼び，図1では ENT-CONS 文，REL-ENT-CONS 文，VALUE-CONS 文で定めている．なお，情報構造は対象世界に忠実に定め，基本項目型や指標はたかだか一つの属性に対応づけるものとする．

（3） アクセス層

入出力データを格納したファイルのアクセス方法を記述する．図1では ACCESS-LAYER 文に続けて記述している．本論文では固定長レコード順呼び出しファイルを入出力するプログラムを対象にして述べる．

ファイルについては名前やレコード長をデータ集合型で定め，図1の DATASET-TYPE [FILE] 文で記述する．レコード長はバイト数で定める．この文の INPUT と OUTPUT は入力と出力の区別を示すための入出力制約である．データ集合型とデータ層のデータ形式を関係づけるため，データ型の木構造を代表する [ROOT] つきのデータ型，たとえば SHOHIN-DATA を文の末尾に記述する．

3.2 実現仕様

プログラム生成の処理対象を脈絡不一致に限定した実現仕様を定めるため，以下の制約を設ける．各制約の効用は次章で述べる．

（1） 主体数制約

主体型に属する主体の最大数を定める．図1の ENT-NUMBER 文は主体型 “商品” に最大 100 個の主体が属することを定めている．最大主体数が不定ならば，主体型 “売上げ” のように ENT-NUMBER 文を省略する．

（2） 主体対応制約

関連型で対応づけられた主体型について主体数の対応関係を CARDINALITY 文で定める．この文には COLLECTION 文に記述した主体型の順序で，その主体型の一つの主体に係わる関連の個数を以下のように記述する．

まず，一つの主体に係わる関連の個数が確定していればその数値を記述する．また，個数がゼロないし1ならば “B”，個数が1以上ならば “I”，個数がゼロ以上ならば “M” と記述する．たとえば，図1の関連型 “売物” では一つの主体 “商品” にゼロ個以上の関連が係わり，一つの主体 “売上げ” に1個の関連が係わる．すなわち，一つの商品を何件もの売上げで取り扱い，一つの売上げは一つの商品しか取り扱わない．

3.3 構造不一致の影響

PSDM のデータ層においては，すべての入出力データを一意な基本項目型名と指標で，制約から直接参照でき設定できる．さらに，情報層ではすべての主体の属性値を役割名と主体型名，属性名，主キー値で，制約から直接参照でき設定できる．したがって，利用者はデータ入力の同期性を考慮しなくてよく，構造不一致は意識せずに言語を利用できる．

次は，情報層へ記述した情報構造によって言語の理解性や記述性の問題が緩和されることを述べる．まず，MODEL で問題視された添字に相当するものは情報層にない．添字に相当する指標はデータ層に現れるが，その使用は単にデータ層と情報層の対応づけを定めるための制約に限定されている．

また，図1の 18 行目と 26 行目の二つの情報制約がともに主体型 “商品” に属する主体の存在を定めているように，3.1 節で述べた主体に関する入力データ用の情報制約，主体と関連に関する入力データ用の情報制約，主体存在従属性制約のいくつかがともに，ある主体型に属する主体の存在を定めているとしよう．この場合，各制約から得られた主体の集合について和集合をとったものがその主体型に存在する．このため，いくつかの制約から同じ主キー値を持つ主体が得られると，その主キー値を持つ主体は一つしか存在しないように自動的に取り扱われる．一方，MODEL では 2.3 節で示した計算式の例からわかるように，主キー値を照合するための条件式を利用者が記述しなければならない．

なお，PSDM では上記のように和集合がとられるので，単一代入の概念に基づいたものとはいえない．

4. プログラム生成方法

4.1 PSDM 適用の制限事項

おもに脈絡不一致が純粋な形で起きるように PSDM へ以下の制限を加えて，プログラム生成方法

を説明する．1) 関連型は二つの異なる主体型を対応づける．2) 主体存在従属性制約を定めた関連型では，主体対応制約を "CARDINALITY 1, 1;" とする．3) 一つの関連型に対して関連存在従属性制約および，主体と関連に関する情報制約の両方を定めてはならない．4) 一つのデータ集合型において入出力レコードに対応するデータ型は一つとする．ただし，入力データの終了を示すための END レコードは別途定めてよい．5) 繰り返し集団項目型は入出力レコードの繰り返しを定めるのに用いる．6) 一つの入出力レコードは同じ主体型について一つの主体しか表さず，同じ関連型についても一つの関連しか表さない．7) 繰り返し制約では同一レコード上のデータ，指標値，または主体型に属する主体の数により条件を定める．選択制約では同一レコード上のデータにより条件を定める．8) 入力ファイルはソートしていない．9) 主体に関する情報制約や，主体と関連に関する情報制約で主キーに対応づけた基本項目型について，同じ主キー値を表している入力データが二つ以上現れてはならない．ただし，REL-ENT-CONS 文の WHERE の右辺に記述したものはこの限りでない．

4.2 構造不一致の検出と解決

MODEL では 2.3 節で述べたように，配列ごとに構造不一致の存否が決まり，不一致が存在する配列に対して主記憶上のテーブルが生成され，不一致が解決される．PSDM でも主体型と関連型ごとに構造不一致の存否を決め，不一致が存在する主体型と関連型に対して主記憶上のテーブルを生成し，不一致を解決するものとする．そこで，以下のようにプログラム仕様を反映した有向グラフを作成し解析して，構造不一致が存在する主体型をまず検出する．なお，本節と次節では図1の仕様からプログラムを生成する過程を例として示す．

（1） 有向グラフの作成

有向グラフは節点と有向枝からなる．節点は表1の仕様形式欄に示すデータ型節点と計算型節点に分けた6種類がある．ここで，主キー属性は主体型節点が代表し，データ型はデータ集合型節点が代表する．一つのデータ集合型に対して情報制約をいくつ定めても，まとめて一つの情報制約節点で表す．関連存在従属性制約はグラフに表さず，次項の局所的な解析へ反映する．繰り返し制約と選択制約については，構造不一致が生じないように4.1節で制限したので，グラフに反映しない．なお，仕様形式はグラフを解析するのに用いる．

有向枝は計算型節点の3種の制約に基づいて，データ型節点と計算型節点のあいだに張る．有向枝の方向はデータの流れが入力用のデータ集合型から出る方向，および出力用のデータ集合型へ入る方向に合わせる．

以上の方法によって図1のプログラム仕様から図2の有向グラフが得られる．ところで，表1の生成形式は主記憶上にテーブルを生成する必要性などを示すものであり，次の(2)，(3)項で述べるように構造不一致の存否が決まれば生成形式も決まる．

（2） 局所的な解析

最初は局所的にプログラム仕様を解析する．ある主体型について3.3節で述べた和集合をとるには，その主体型に属する主体の主キー値を主記憶上のテーブルへ蓄積し，そのテーブル上で同じ主キー値を持つ主体を検出するための照合が必要となる．このため，テーブルへ書き込まれた個々の主キー値は交互に何回も参照されるので，書き込みと参照が同期せず，その主体型に構造不一致が存在する．このように蓄積と照合が

表 1 有向グラフの節点
Table 1 Nodes of directed graph.

仕様形式 ＼ 生成形式		蓄積型節点	蓄積照合型節点	書換え型節点
データ型節点	主体型節点	○	○	○
	非主キー属性節点	○	×	○
	データ集合型節点	○	×	×
計算型節点	属性値従属性制約節点	×	×	○
	主体存在従属性制約節点	×	×	○
	情報制約節点	×	×	○

○ 仕様形式と生成形式の組合せあり．× 左記の組合せなし．

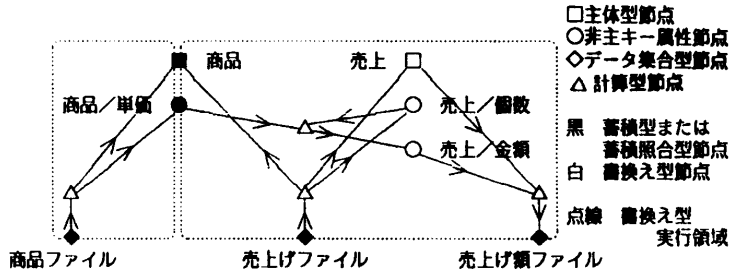

図 2 有向グラフと書換え型実行領域
Fig. 2 A directed graph and execution regions with rewriting type.

必要な主体型の節点は，生成形式を蓄積照合型とする．図2では商品が蓄積照合型となる．

さて，属性値従属性制約が値を参照する相手の主体型について，その一つの主体にいくつもの関連が係わるように主体対応制約を定めたとしよう．この場合，属性値従属性制約で定めた計算が同一主体の同じ属性値を何回も参照する．たとえば，同一商品の単価は何回も参照される．しかし，その商品を取り扱った売上げが現れたときに参照される．このため，参照される属性値を主記憶上のテーブルへ蓄積し，いつでも参照できるようにする．このテーブルへの書き込みと参照も個々の属性値について見ると同期しないので，その主体型に構造不一致が存在する．このように蓄積のみが必要な主体型の節点は，生成形式を蓄積型とする．この観点から見ると，図2の商品は蓄積型にもなる．

一方，属性値従属性制約が値を設定する相手の主体型について，その一つの主体にいくつもの関連が係わるように主体対応制約を定めたとしよう．これは総和のように不定個数のパラメータをとる関数に必要である．このような関数では中間結果がとられるが，その中間結果は，値が設定される主体型の主体ごとに存在する．しかも，ある主体の中間結果は，その主体に対応するパラメータの属性値が現れるつど何回も更新される．このため，中間結果は，値が設定される主体型の主体に対応づけて主記憶上のテーブルへ蓄積し，いつでも更新できるようにする．このテーブルの更新も個々の中間結果について見ると同期しないので，その主体型に構造不一致が存在する．その主体型の節点も生成形式を蓄積型とする．

関連存在従属性制約では，関連型で対応づけられた二つの主体型から一つずつとった主体のすべての組合せについて，関連の存否を判定する．このため，同一主体の同じ属性値が交互に何回も参照される．このため，その主体型の全主体の属性値を主記憶上のテーブルへ蓄積し，同じ属性値を交互に何回も参照できるようにする．このテーブルへの書き込みと参照も個々の属性値について見ると同期しないので，その主体型に構造不一致が存在する．その主体型の節点も生成形式を蓄積型とする．

データ集合型はデータを蓄積したものなので，その節点も生成形式を蓄積型とする．図2の商品と売上げ，売上げ額の三つのファイルは蓄積型である．

以上の方法によって主体型節点は蓄積型か蓄積照合型になるが，前述の商品のように両方の生成形式が決まることもある．この場合は蓄積照合型とする．一方，蓄積型でもなく蓄積照合型でもない主体型には，局所的な解析において構造不一致が存在しないものと見なし，生成形式を書換え型とする．構造不一致が存在しない主体型に対してはテーブルの生成が不要であり，一つの主体の属性値のみを保持できるデータ域を主記憶上に生成し，それを主体ごとに書き換えて用いる．このデータ域への書き込みと参照は同期する．図2では売上げが書換え型となる．なお，次項の大局的な解析において書換え型から蓄積型へ変わる主体型が現れる．

ところで，蓄積型か蓄積照合型と決まった主体型の非主キー属性の節点は生成形式を蓄積型とし，その他の非主キー属性節点は書換え型とする．図2では商品／単価が蓄積型となり，売上／個数と売上／金額は書換え型となる．計算型節点にはデータを蓄積せず，テーブルは不要なので，生成形式を書換え型とする．

（3） 大局的な解析

有向グラフの有向枝をつないで有向道ができる．なお，有向道がループするとプログラムの生成に特別な工夫がいるので，これを避けるため，PSDM では有向道がループしないようにプログラム仕様を記述する．

さて，任意の二つの節点間の有向道が書換え型節点のみを通れば，その道を書換え型有向道と呼び，蓄積型節点か蓄積照合型節点を一つでも通れば蓄積型有向道と呼ぶ．なお，道の始点と終点の節点については生成形式を問わない．ところで，書換え型のデータ型節点へデータを書き込むと，その節点では参照が書き込みと同期するので，その節点の次にある計算型節点の処理を直ちに実行できる．この性質を利用して，書換え型有向道上では，道の始点から一つのデータを受け取るつど，道にそって書換え型節点を経由しながらデータを処理し，道の終点へデータを渡すものとする．

一方，蓄積型節点や蓄積照合型節点へデータを書き込んでも，その節点では参照が書き込みと同期しないので，その節点の次にある計算型節点の処理を直ちに実行できるわけではない．蓄積型有向道上では始点から受け取ったデータを処理し，その道の途中にある蓄積型節点か蓄積照合型節点へデータを蓄積するが，前述の理由により，全データの蓄積が終わらなければ次の計算型節点の処理を実行しないものとする．すなわち，始点から全データを受け取った後でなければ，終

点へデータを渡すことができない.

大局的な解析においては，書換え型節点で分岐し，途中で交わることなく計算型節点で合流する有向道群に着目する．ところで，書換え型節点ではデータの書き込みと参照が同期するので，群のなかの各々の道も同期して分岐点の書換え型節点からデータを受け取るものとする．しかし，その群に書換え型有向道と蓄積型有向道が混在すると，書換え型有向道上では分岐点からデータを受け取るつど合流点へデータを渡すが，蓄積型有向道上では分岐点から全データを受け取った後でなければ合流点へデータを渡すことができない.

その結果，合流点では書換え型有向道から渡されるデータと，蓄積型有向道から渡されるデータが同期しない．しかし，合流点の計算型節点は書換え型でありデータを蓄積しないので，その節点へデータを同期して渡さなければならない．すなわち，その節点に構造不一致が存在する．そこで，書換え型有向道上で任意のデータ型節点を書換え型から蓄積型へ変え，その節点でデータを蓄積して待ち合わせる．それで同期をとり不一致を解決する．このとき，その節点と同じ主体型に属する書換え型節点もすべて蓄積型へ変える.

図2では売上げファイルと売上げ額ファイルの二つの情報制約節点のあいだに有向道群が存在するが，群のなかの道はすべて書換え型なので生成形式に変化はない．したがって，売上げは書換え型のままである.

4.3 プログラムの生成

（1） データ域の生成

前節の方法によって主体型の生成形式が決まるので，その生成形式に基づいて以下のようにデータ域を生成する．主体型の生成形式が蓄積型か蓄積照合型であれば，その主体型に属する全主体の属性値を主記憶上に蓄積するため，テーブルを配列変数で生成する．配列の繰り返し数は主体数制約で与えられた主体数に合わせる．一方，生成形式が書換え型であれば，一つの主体の属性値を一時保持するためのスカラ変数を生成する．図2では商品に対してテーブルが配列変数で生成され，その配列の繰り返し数は 100 となる．一方，売上げに対してはスカラ変数が生成される.

関連型については生成形式をまず決める．関連型で対応づけられた二つの主体型がともに蓄積型であれば，その関連型も蓄積型とする．二つの主体型がともに蓄積照合型であるか，あるいは蓄積型と蓄積照合型の組合せであれば，関連型も蓄積照合型とする．その他の関連型は非生成型とする.

蓄積型か蓄積照合型の関連型に対して，主記憶上のテーブルを配列変数で生成するのは主体型と同じである．しかし，関連型で対応づけられた二つの主体型に対して生成された配列の添字値の組を，関連としてテーブルへ蓄積する．配列の繰り返し数は主体対応制約と，二つの主体型の主体数制約から計算して決める．たとえば，主体対応制約がともにMであり，主体数制約が 100 と 5 であれば，繰り返し数を 500 (100×5) とする.

一方，非生成型に対してはデータ域を生成しない．この場合，関連型で対応づけられた二つの主体型について，関連を持った二つの主体が同時に処理対象となるように手続きを生成する．図2では，蓄積照合型の商品と，書換え型の売上げを対応づけた関連型“売物”は非生成型になるので，データ域は生成されない.

アクセス層のデータ集合型については各々1レコード分のデータ入出力域を生成する．データ層のデータ型に対してはデータ域を生成しない.

（2） 手続きの生成

前項の方法によってテーブルが生成されるので，テーブルへデータを蓄積し終われば，そのテーブルを参照する処理が実行されるように手続きを生成する.

まず，有向グラフを書換え型実行領域と呼ぶ部分グラフに分割する．この領域は，始点と終点が蓄積型か蓄積照合型である書換え型有向道上の節点と有向枝からなり，しかも連結グラフとする．すなわち，書換え型有向道がその途中の節点で分岐か合流することによって，領域中の全節点が，方向を無視した道でつながっているものとする．さらに，領域の境界の節点(以後，境界点という）は蓄積型か蓄積照合型であるが，領域の内部の節点はすべて書換え型である．このため，領域の内部ではデータの流れが同期し，構造不一致は生じない．なお，いくつかの領域が境界点として蓄積型節点か蓄積照合型節点を共有できるが，書換え型節点と有向枝は共有しないものとする．図2では点線のように二つの領域に分かれる.

手続きは書換え型実行領域ごとに以下のように生成する．一つの領域から生成される手続きは一つの制御ループからなる．このループのなかでは最初に，流出有向枝を持つ境界点に対して生成されたテーブルから，一つの主体が持つ属性値を参照する．境界点がデータ集合型であれば，一つのレコードを入力する．次に，その属性値やレコードについて各々の書換え型

有向道上の制約で定められた計算を順に実行する．最後に，流入有向枝を持つ境界点に対して生成された各テーブルへ，一つの主体が持つ属性値を蓄積する．境界点がデータ集合型であれば，一つのレコードを出力する．

ここで，流出有向枝を持つ境界点については，ループの繰り返しごとにテーブルの先頭の主体から順に属性値を参照し，すべて参照してしまえばループの繰り返しが終わる．データ集合型の場合，レコードの繰り返し制約が成り立てば，ループの繰り返しが終わる．

蓄積照合型の主体型節点のテーブルへ主体の属性値を蓄積する場合，すでにテーブルへ蓄積されている主キー値と照合し，同じ主キー値がなければ実際に主キー値や非主キー属性値をテーブルへ書き込む．一方，同じ主キー値があれば，その主キー値と同じテーブルの行へ非主キー属性値のみを実際に書き込む．また，図2の右側の領域において商品／単価を参照するような場合も，照合が成立した主キー値と同じテーブルの行から非主キー属性値を参照する．

関連型の生成形式が蓄積型か蓄積照合型であれば，その関連型で定められた関連存在従属性制約や，主体と関連に関する入力データ用の情報制約を一つずつ処理し，その関連型に対して生成されたテーブルへ関連を蓄積する．特に蓄積照合型の場合，同じ関連がすでに蓄積されていないか照合して蓄積する．この蓄積が終われば，その関連型で定められた属性値従属性制約や，主体と関連に関する出力データ用の情報制約を，テーブルに蓄積された関連ごとに繰り返して処理する．

以上の方法によって，図2の左側の領域では，商品ファイルからレコードを入力し，そのなかの品名と単価を商品テーブルへ蓄積するための手続きが生成される．右側の領域では以下の手続きが生成される．まず，売上げファイルからレコードを入力し，そのなかの品名と，商品テーブル中の品名を照合して単価を得る．次に，単価と個数の積をとって金額を得て，売上げ額ファイルへレコードを出力する．

書換え型実行領域が2個以上あれば，流出有向枝を持つ境界点が各々，以下のいずれかの条件を満たした領域から順に実行する．1) 境界点が入力用のデータ集合型節点である．2) 他の領域の実行によって，境界点のテーブルに属性値の蓄積が終了した．図2では左側の領域が先に実行され，次に右側の領域が実行される．

4.4 実　　験

前節までに述べた方法を適用して，実験用のCOBOLプログラム・ジェネレータを作成した．ただし，実験を容易にするため，おもに以下の制限を加えた．1) 有向グラフ上で属性値従属性制約節点と主体存在従属性制約節点は，その節点の流出有向枝で直接結ばれたデータ型節点と同一視し，また情報制約節点はその節点と有向枝で直接結ばれたデータ型節点と同一視した．2) 属性値を入れるための配列変数やスカラ変数のデータ形式は一律に，数値を“PICTURE S9 (10) V9 (5).”とし，文字列を“PICTURE X (16).”とした．なお，ジェネレータはCOBOLで作成し，その規模は7Kステップであった．ただし，文法の検査機能は実現しなかった．

実験には，PSDL-2による記述が100ステップ以下のプログラム仕様を適用した．その結果，生成されたCOBOLプログラムの静的ステップ数は，プログラム仕様のステップ数の約4倍であった．また，手書きのCOBOLプログラムの静的ステップ数と比較して2-3倍であった．

ところで，本論文に述べた構造不一致の検出方法には，脈絡不一致に限定しても，まだ検出精度の向上余地が残っている．たとえば，有向道群を用いた大局的な解析方法を拡張して，さらに広い範囲の構造不一致を検出することなどがあげられる．しかし，この精度向上を図っていけば，上記のステップ数比較から見て，実用的な性能のプログラムを生成することが可能になるものと予想される．

5. ま と め

入出力データのあいだの構造不一致が非手続き型言語へ及ぼす影響について述べ，PSDMにおける構造不一致の取り扱いについて報告した．PSDM用に考案した本論文のプログラム生成方法は構造不一致の検出対象がまだ脈絡不一致に限定され，検出精度に向上の余地も残っているが，生成されたCOBOLプログラムの静的ステップ数は手書きのCOBOLプログラムと比較して2-3倍であった．このため，実用的な性能のプログラムを生成することが可能になるものと予想される．

今後は構造不一致検出精度の向上を図るとともに，ソートされたファイルや，複雑な入出力データ形式，乱呼び出しファイル，構造不一致解決方法の多様化，生成過程への利用者介入なども含めてプログラム生成

方法を研究する予定である．

謝辞 本研究はNTT情報通信処理研究所で行ったものであり，本研究をご指導ご支援いただいた伊吹公夫もと特別研究室長（現東京工科大学工学部教授）に厚くお礼申しあげる．

参考文献

1) Jackson, M. A.: *Principles of Program Design*, p. 299, Academic Press, London (1975) (鳥居宏次訳：構造的プログラム設計の原理，p. 318，日本コンピュータ協会，東京 (1980))．
2) Hoffmann, C. M.: Design and Correctness of a Compiler for a Non-Procedural Language, *Acta Informatica*, Vol. 9, pp. 217-241 (1978)．
3) Ashcroft, E. and Wadge, B.: Some Common Misconceptions about Lucid, *ACM SIGPLAN Notices*, Vol. 15, No. 10, pp. 15-26 (1980)．
4) Prywes, N. S. and Pnueli, A.: Compilation of Nonprocedural Specifications into Computer Programs, *IEEE Trans. Softw. Eng.*, Vol. SE-9, No. 3, pp. 267-279 (1983)．
5) 橋本正明：プログラム仕様記述のための計算指向 EAR モデル，情報処理学会論文誌，Vol. 27, No. 3, pp. 330-338 (1986)．
6) 橋本正明：EAR モデルに基づく情報構造記述を用いたプログラム仕様記述法 PSDM，情報処理学会論文誌，Vol. 27, No. 7, pp. 697-706 (1986)．
7) 橋本正明：データ中心のプログラム仕様記述法，p. 210，井上書院，東京 (1988)．
8) Tesler, L. G. and Enea, H. J.: A Language Design for Concurrent Processes, *Proc. of SJCC*, pp. 403-408 (1968)．

(昭和63年 3 月 10 日受付)
(昭和63年10月 7 日採録)

橋本 正明（正会員）

昭和21年生．昭和43年九州大学工学部電子工学科卒業．昭和45年同大学院修士課程修了．同年日本電信電話公社電気通信研究所勤務．昭和63年 ATR 通信システム研究所出向．これまで主にオペレーティング・システム，中間言語マシン，データベース設計支援ツール，ソフトウェア自動作成の研究実用化に従事．著書「データ中心のプログラム仕様記述法」（井上書院）．工学博士．電子情報通信学会会員．