プログラミング言語　25－3
（１９９０．６．８）

非手続き型言語の集合と写像の性質に着目した入出力データの構造不一致検出・解決法

橋本正明　　岡本克己
(株) ATR 通信システム研究所

非手続き型言語の集合と写像の性質に着目して,JSP 法の主テーマである入出力データの構造不一致を自動的に検出して解決する方法とその実験について述べる. ところで, 構造不一致はプログラムの仕様に係わる問題ではないが, 効率のよいプログラムを作成するには構造不一致を検出して解決しなければならないので, 構造不一致をプログラマが意識しなくてもよく, コンパイラが検出して解決してくれるという重要な非手続き型言語のクラスが存在する. 集合と写像は高位の非手続き型言語に採用される傾向にあるので, 上記のクラスでそれらの言語を実現するのに, 本稿の構造不一致検出・解決法が参考になるものと予想される.

A Set and Mapping-based Detection and Solution Method for Structure Clash between Program Input and Output Data in Nonprocedural Languages

Masaaki HASHIMOTO and Katsumi OKAMOTO
ATR Communication Systems Research Laboratories
Sanpeidani, Inuidani, Seika-cho, Soraku-gun, Kyoto 619-02, Japan

This paper describes a set and mapping-based automatic detection and solution method for structure clash between program input and output data. Structure clash is one of the main concerns in JSP. Structure clash is not a program specification issue, and must be detected and solved in order to produce a efficient program. Therefore, there is a very important nonprocedural language class in which a programmer need not think about the clashes, but in which a compiler detects and solves the clashes. Sets and mappings often appear in high-level nonprocedural languages. The method will be useful for the implementation of such languages in the class.

1 はじめに

プログラムの入出力データの構造不一致は, プログラムで同時に処理される入出力データの間で入出力タイミングが同期していない問題として定義されている.[1] そこで,C 言語のような手続き型言語のプログラマが, 構造不一致の存在するプログラムを作成する場合, その構造不一致を検出しなければならない. それから, 構造不一致を解決するため, データをプログラム内のテーブル等に保持して待ち合わせることによって, 同期がとれるようにプログラムを作成しなければならない.

ところで, 構造不一致の検出や解決はプログラムの仕様に係わる問題ではなく, プログラムの実現に係わる問題である. しかも, メモリ空間や実行時間の効率がよいプログラムを作成するには, 構造不一致を精度よく検出して解決しなければならない. このため, プログラマは構造不一致を意識しなくてもよく, コンパイラが構造不一致を検出して解決してくれるという重要な非手続き型言語のクラスが存在する. そのクラスには配列の性質に着目して構造不一致を検出して解決する MODEL 言語 [2] が属している.

筆者らはそのクラスにおいて実体関連モデル (Entity Relationship Model) を適用した非手続き型言語 PSDL(Program Specification Description Language) を研究している.[3]-[5] ところで, 実体型または関連型は各々実体または関連の集合であり, さらに関連型は実体の間の写像でもある. このように,PSDL は上記の配列とはまったく異なった集合と写像という性質を備えている. 集合と写像は高位の非手続き型言語に採用される傾向にあるので, 筆者らはそれらの性質に着目した構造不一致検出・解決法を研究した.[6]-[8]

本稿の目的はこの検出・解決法とともにその実験について報告することである. なお, 本稿では構造不一致のうち, 脈絡不一致と呼ばれているもの [1] の部分問題を取り扱う. また, 文献 [6] と比較して, 本稿では同期性解析対象の変更や, 大局的な解析方法の一般化によって検出と解決の精度が向上した. さらに, プログラムの大局的最適化の問題を具体化できた. 第 2 章では PSDL を概説し, 第 3 章で構造不一致を説明する. 第 4 章と第 5 章では構造不一致の検出法と解決法を述べる. 第 6 章で実験について述べ, 第 7 章で考察する.

2 PSDL

PSDL ではプログラムの入出力データの性質に着目して, プログラム仕様を以下の 3 階層に分けて記述する.
a) プログラムの入出力データに表される対象世界の情報について, その枠組を定めた情報層.
b) 帳票のような入出力データ形式を定めたデータ層.
c) 入出力ファイルのアクセス方法を定めたアクセス層.

本稿の説明に用いられるプログラム仕様を記述した図 1 と, その仕様を図示した図 2 を用いて PSDL 文を概説する. このプログラムはファイル product_file と sale_file を入力して, 売上毎に計算した売上額をファイル account_file へ出力する. その時, 顧客毎に計算した売上合計も併記して出力するものとする.

2.1 情報層

情報層は, 図 1 では 00 行目の INFORMATION 文と 26 行目の DATA 文の間に記述し, 図 2 では上半分の点線の四角形内に図示している.

(1) 実体型, 属性, 主キー, 実体数

入出力データは図 2 の Television や, Sale No. 1, Mr. Tanaka というような実体を表している. これらの実体の集合である実体型 product や, sale, customer は太線の四角形で図示する. 実体型の主キー属性と非主キー属性はその四角形の下に示している.

図 1 では各々の実体型を 01 行目の E (Entity type) 文で記述する. それに続けて主キー属性を 03 行目の K (Key) 文で記述し, 非主キー属性を 04 行目の A (Attribute) 文で記述する. 03 行目の STR (STRing) は属性値の定義域が文字列であることを示し,04 行目の NUM (NUMber) は数値であることを示す. 実体型に含まれる実体数は 02 行目の EN (Entity Number) 文で記述する. 02 行目の "-50" は実体型 product に含まれる実体が最大 50 個であることを示している.

(2) 関連型, 実体型の対応づけ, 関連数

入出力データは図 2 の点線の直線で示した "Televisions are *sold* in sale No. 1." や "Mr. Tanaka *buys* in sale No. 1." というような関連も表している. これらの関連の集合である関連型 sold や buy は太線の菱形で図示している.

図 1 では各々の関連型を 05 行目の R (Relationship type) 文で記述する. それに続けて, 関連型で対応づけられた実体型を 06 行目と 08 行目の C (Collection) 文で

```
00 INFORMATION
01   E product
02       EN -50
03     K name   STR
04     A price NUM
05   R sold
06     C . product
07       RN M
08     C . sale
09       RN 1
10   E sale
11       EN -100
12     K number    NUM
13     A quantity NUM
14     A amount    NUM
15       = . sold. . product. price * quantity
16   R buy
17     C . sale
18       RN 1
19     C . customer
20       RN M
21   E customer
22       EN -50
23     K name   STR
24     A total NUM
25       = SUM(. buy. . sale. amount)
26 DATA
27   I product_data
28         IX product_index
29       G product_record ON ENDOFFILE(product_file)
30     Q product_record
31       %12s product_name
32         = product. name
33       %8d  product_price
34         = product. price
35   I sale_data
36         IX sale_index
37       G sale_record ON ENDOFFILE(sale_file)
38     Q sale_record
39       %4d  sale_number
40         = sold. . sale. number
41         = buy. . sale. number
42       %16s sale_customer
43         = buy. . customer. name
44       %12s sale_product
45         = sold. . product. name
46       %4d  sale_quantity
47         = sale. quantity
48   I account_data
49         IX account_index
50       G account_record ON ENTITYNUMBER(sale)
51     Q account_record
52       %4d  sale_number
53         = sale. number
54         = buy. . sale. number
55       %8d sale_amount
56         = sale. amount
57       %16s sale_customer
58         = buy. . customer. name
59       %10d customer_total
60         = customer. total
61 ACCESS
62   D product_file INPUT  20 product_data
63   D sale_file    INPUT  36 sale_data
64   D account_file OUTPUT 38 account_data
```

図1 PSDL プログラム仕様

指定する. この文には実体型とその役割を Role.Entity Type の形で指定する. それらの実体型が相互に異なる場合は図1のように役割を省略してよい. 実体型を指定したC文の後には, その実体型の1つの実体につながる関連の個数を 07 行目の RN (Relationship Number) 文で記述する. 07 行目の M (Many) は個数が不定であることを示し,09 行目の 1 は 1 個であることを示している.

(3) 属性値従属性制約

この制約は非主キー属性の値を得るために用いる. 図2の矢印で示すように実体 sale の属性 amount の値は, その実体へ関連 sold で対応づけられた実体 product について, その属性 price の値と sale の属性 quantity の値を掛けて得られる. また, 実体 customer の属性 total の値は, その実体へ関連 buy で対応づけられた零個以上の実体 sale について, その属性 amount の値を合計して得られる.

図1では, 値が得られる非主キー属性の A 文に続けて,15 行目の = (equal) 文で制約を記述する. この制約から参照される属性は attribute または role1.relationship Type.role2.entityType.attribute の形で記述する. ここで前者は, 値の得られる実体が持っている他の属性を参照するのに用い, 一方その実体へ関連で対応づけられた他の実体の属性を参照するのに後者を用いる. なお, 前者のみが現れる制約も記述してよい. また, 後者を用いる場合,1 つの制約に 1 つの関連型しか記述できない. ところで, 上記の role1 は値が得られる方の実体の役割を示し,role2 は他方の実体の役割を示している.

(4) 関連存在従属性制約

この制約は関連を得るのに用いる. この例は図1にないので以下に示す. たとえば, 実体 person が持っている属性 skill の値が, 実体 section に適した属性 skill の値に等しければ, その場合にのみ両実体が関連 belong_to で対応づけられるものとする. この関連存在条件は R 文と C 文に続けて,RC (Relationship existence Condition) 文で "RC (.person.skill == .section.skill)" と記述する. ここで, 条件式から参照される属性は role.entity Type.attribute の形で記述している.

(5) 実体存在従属性制約

この制約は実体を得るのに用いる. この例も図1にないので以下に示す. たとえば, 実体 customer の属性 total が正値であれば, その場合にのみ, その実体に関連 demand で対応づけられる実体 account が存在するものとする. この時, 得られた実体の主キー属性値を決めなければならないので, 実体が得られる実体型の K 文に続けて, 主キー属性値を決めるための計算式を属性値従属性制約と同じ = 文で "= .demand..customer.name ON (.demand..customer.total > 0)" と記述する. この

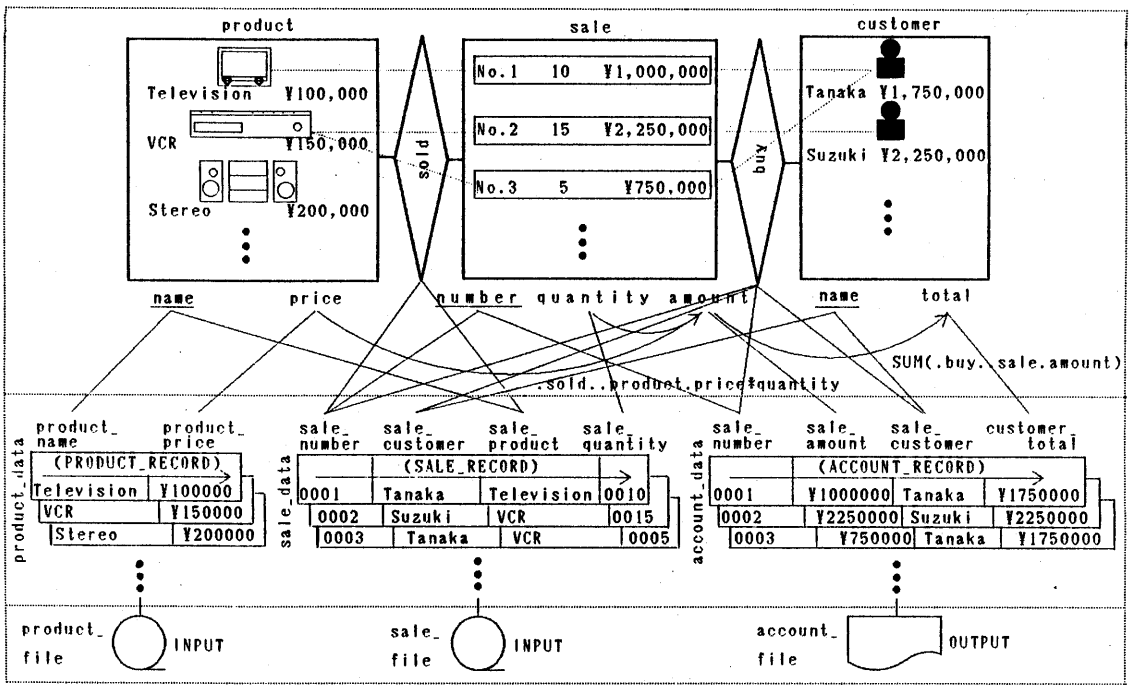


図 2 PSDL プログラム仕様図

例では account の主キー属性値は customer の主キー属性 name の値に等しいものとした. また,total の値が正か否かを判定するための条件式は ON 句で記述する. なお,1 つの制約に 1 つの関連型しか記述できない.

2.2 データ層とアクセス層

(1) データ層

データ層は, 図 1 では 26 行目の DATA 文と 61 行目の ACCESS 文の間に記述し, 図 2 では中段の点線の四角形の中に示している. 入出力データの構造は基本データ型や, 連接集団データ型, 繰返し集団データ型, 選択集団データ型で階層的に定める. これらのデータ型は各々 % 文や, Q (seQuence) 文, I (Iteration) 文,S (Selection) 文で記述する.

% 文は基本データ型のデータ形式も C 言語と同様に定める. たとえば,%12s は 12 文字の文字列を定めている. I 文には続けて指標も IX 文で記述する. 上記の集団データ型は他のデータ型から構成されるが, 構成要素のデータ型の中に集団データ型があれば, それを G (Group) 文で指定する. 繰返し集団データ型には繰返し終了条件が必要であり,29 行目の ON 句は End Of File 条件を示し,50 行目の ON 句は実体 sale の個数だけ繰り返すことを示している.

ところで, 本稿では入出力データは実体や, その属性値, 関連を表すものと見なしているので, まず実体については実体型の主キー属性を図 1 の 32 行目の = 文で基本データ型と結合する. 属性値については属性を 34 行目の = 文で基本データ型と結合する. 関連については, 関連で対応づけられた実体型の主キー属性を 40 行目と 45 行目の = 文で基本データ型と結合する. なお, 実体のないところには関連もないので, 入出力データには関連とともに実体も表される. 上記の結合をとることは情報層とデータ層の結合制約と呼ぶ.

(2) アクセス層

図 1 の 61 行目の ACCESS 文に続けて各々のファイルをデータセット型として 62 行目の D (DatasetType) 文で記述する. この文にはファイル名や,INPUT と OUTPUT の区別, レコード長を記述する. データ層とアクセス層の結合制約として product_data のようにデータ型を指定する.

3 構造不一致

入出力データの構造不一致について説明する.

(1) 構造不一致の例

前章で説明したプログラムには, 入力レコードがソー

トされていないという前提のもとで構造不一致が存在している. すなわち,sale amount の計算や,customer total の計算,sale amount と customer total を併記した出力について入出力タイミングが同期していない. これらの構造不一致を解決するには,product name と,product price, sale amount, customer total, sale と customer の対応づけをテーブル等に保持して待ち合わせることによって, 同期がとれるようにプログラムを作成しなければならない.

(2) 構造不一致による非手続き型言語のクラス分け

非手続き型言語は構造不一致の観点から以下のようにクラス分けができる.

クラス 1) プログラマが構造不一致を意識しなければならない. たとえば,Basic Lucid[9] はスカラ変数しか持っておらず, スカラ変数の上に多量のデータを同時に保持するのは困難なので, 構造不一致が存在するプログラムは現実的には記述できない. そこで, プログラマは構造不一致が存在しないことを確認してからプログラムを記述しなければならない.

クラス 2) プログラマは構造不一致を意識しなくてもよい. コンパイラも構造不一致の検出は行わず, 常に構造不一致が存在するものと仮定して, すべてのデータにテーブル等を割り当てて構造不一致を解決する. たとえば, エンドユーザ言語 LOTUS 1-2-3 がこのクラスに属している.

クラス 3) プログラマは構造不一致を意識しなくてもよい. 一方, コンパイラが構造不一致を検出して解決してくれる. このクラスに属する MODEL 言語 [2] では, すべての入出力データと中間データを配列変数で記述する. そこで, 配列毎に構造不一致が存在するか否かを検出して, 構造不一致が存在しない配列にはスカラ変数を割り当て, 一方構造不一致が存在する配列にはテーブルを割り当てて構造不一致を解決している.

上記のクラス 2) と 3) の言語が存在することは, 構造不一致がプログラムの仕様に係わる問題ではなく, プログラムの実現に係わる問題であることを示している. また, データへテーブルを割り当てるよりはスカラ変数を割り当てるほうが, プログラムのメモリ空間や実行時間の効率がよいので, 構造不一致は精度よく検出して解決しなければならない. このため, 筆者らはクラス 3 の言語として PSDL を研究中である.

(3) 構造不一致の統一的な見方

文献 [1] では 3 種の構造不一致, すなわち脈絡不一致と, 順序不一致, 境界不一致が個々に例示的に説明されている. 一方, 構造不一致の統一的な検出方法と解決方法を解明するには, 構造不一致の統一的な見方が必要である. ところで, 構造不一致における同期性は以下の 3 つの要素に依存している.

a) テーブルに保持されたデータや, スカラ変数に代入されたデータのようなタイミングを担う対象.

b) プログラムで同時に処理される対象どうしの組合せ.

c) 同じテーブルに保持されたデータが流れる順序はプログラムの入力順序で決められるが, そのように同じ型を持っている対象の流れの順序.

そこで, 対象の組合せが起きる箇所で, 対象の流れの順序が一致していなければ, その箇所が非同期点となる. ところで, 脈絡不一致は対象どうしの組合せに係わり, 順序不一致は流れの順序に係わり, また境界不一致は対象自体のとり方に係わっているものと考えられる.

4 構造不一致の検出法

本稿では以下の制限のもとで構造不一致の検出法と解決法を述べる.

a) 入出力ファイルは順アクセスされ, 入力レコードはソートされていない.

b) 入出力レコードに相当したデータ型は 1 つのファイルに 1 つしかない.

c) 1 つの入出力レコードは同一の型の実体は高々 1 つしか表さず, 同一の型の関連も高々 1 つしか表さない.

さて, 構造不一致を検出するには, 以下に述べるように,PSDL プログラム仕様から作成された有向グラフの上で同期性を局所的かつ大局的に解析する.

4.1 有向グラフ

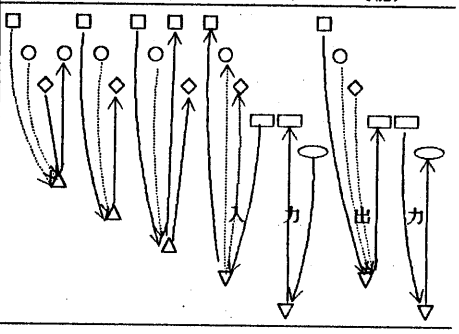
グラフの作成方法と同期性の解析対象を述べる.

(1) 有向グラフの作成方法

グラフの節点は PSDL 文に従って集めるが, 表 1 に示すように実体型節点のような節点を 10 種設け, それらを構造節点と制約節点とに分ける. 実体型節点は実体型に対応するのみでなく, その主キー属性にも対応している. なお, 上記の制限 b),c) を設けたので, データ型節点には繰返し集団データ型のみを反映すればよい. さらに, 繰返し集団データ型毎に情報層とデータ層の結合制約を集約して 1 つの節点で表す.

グラフの有向枝は表 1 に示すように構造節点と制約

表1 節点と有向枝

節点		有向枝 (←――:必須 ←┈┈:可能)
構造節点	実体型節点	
	非主キー属性節点	
	関連型節点	
	データ型節点	
	データセット型節点	
制約節点	属性値従属性制約節点	
	関連存在従属性制約節点	
	実体存在従属性制約節点	
	情報層とデータ層の結合制約節点	
	データ層とアクセス層の結合制約節点	

節点の間に張る. 枝の方向は, 情報層では実体や, 属性値, 関連が制約から参照される方向と, 制約から得られる方向に合わせ, データ層とアクセス層ではデータが入力用データセット型から流出する方向と, 出力用データセット型へ流入する方向に合わせる. たとえば, 図3は図1の仕様から作成されたグラフである. なお, 本稿ではグラフの中に単一方向閉路はないものとする.

(2) 同期性の解析対象

実体や, 属性値, 関連, データが前章の (3) 項で述べたタイミングを担う対象として有向枝に沿って流れており, 同時に処理される対象の組合せは節点の箇所で起きるものと考えられる. ところで, 第2章で述べた制約は, 制約節点で対象の流れの順序が一致するように定式化している. また, 上記の制限 b),c) を設けたので, データ型節点でも流れの順序が一致する. データセット型節点は1本しか枝を持たないので, その節点でも流れの順序が一致するものと見なしてよい.

そこで, 有向枝を非同期型と同期型とに分け, 非同期型有向枝は実体型節点か, 非主キー属性節点, 関連型節点を介してつながった他の有向枝との間で対象の流れの順序が一致せず, 一方任意の節点を介してつながった同期型有向枝の間では対象の流れの順序が一致するものとする.

4.2 局所的な同期性の解析

局所的な解析によって非同期型有向枝を検出する.

(1) 実体や関連の和集合

実体型節点か関連型節点に2本以上の枝が流入していれば, それらの枝を非同期型にする. また, それらの実体型の非主キー属性節点へ流入している枝も非同期型にする. たとえば, 図3では実体型節点 product へ2本の枝が流入しているので, 枝 A1 と, A2, A3 とが非同期型になる. 以下, 実体型節点について理由を説明する.

実体型節点に流入している2本以上の枝の各々から, 同じ主キー値を持っている実体が2個以上流れて来ても, それらは1つの実体と見なされる. このため, 各々の枝から得られた実体の集合に対して和集合をとらなければならない. 和集合をとるには実体の集合の間で主キー値を照合しなければならないが, 入力データはソートされていないので, 同じ主キー値を持った実体が各々の流入枝から同期して流れて来るとは限らない. そこで, すべての実体を待ち合わせて照合しなければならないので, すべての実体が流れ込んで和集合演算が終了した後, 実体の属性値を参照できるものとする. このため, 上記の実体型節点と非主キー属性節点へ流入している枝の上の流れは, それらの節点を介してつながった他の枝の上の流れと同期しない.

(2) 写像の数量関係

実体型節点または非主キー属性節点と, 属性値従属性制約節点または実体存在従属性制約節点との間にある枝は, 実体相互の写像の数量関係を表す RN 文の関連数と対応している. その関連数が2以上か M であれば, その枝を非同期型にする. その理由を以下に説明する. なお, 関連存在従属性制約については, 関連型で対応づけられた実体型どうしの直積集合の全要素へ, 関連存在条件を適用しなければならない. このため, 同じ実体が2個以上の要素の中に出現しうるので関連数を仮想的に M と見なして, 上記と同じ方法で同期性を解析する.

a) 上記の3種の制約が属性値を参照する枝の関連数が2以上か M の場合, 同じ属性値が何回も参照される. ところが, 入力データはソートされていないので, 同じ属性値が連続して参照されるとは限らない. そこで, 属性

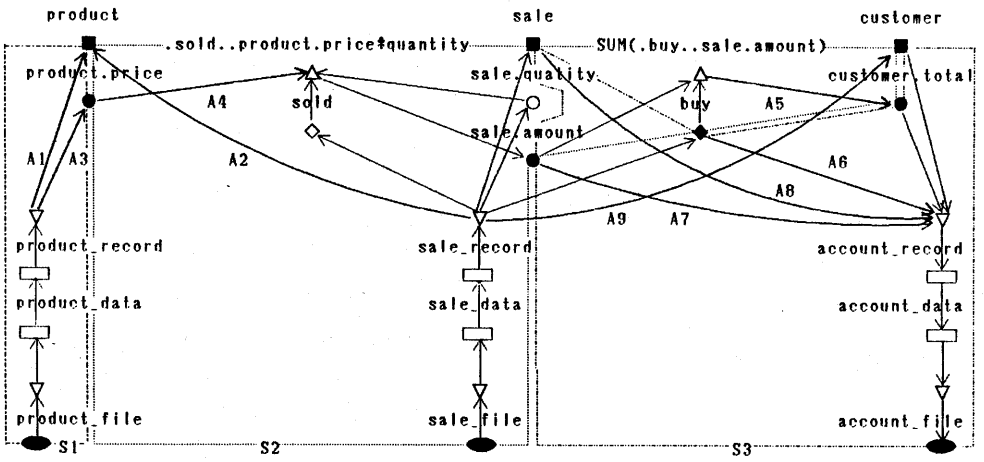


図3 有向グラフ

値がいつ参照されてもよいように待ち合わせなければならないので, すべての属性値が流れ込んだ後, その枝から参照できるものとする. このため, その枝の上の流れは, 実体型節点か非主キー属性節点を介してつながった他の枝の上の流れと同期しない. たとえば, 図3では枝A4が関連数Mを持つので非同期型になる.

b) 属性値従属性制約によって非主キー属性値が得られる枝の関連数が2以上かMの場合, その制約には図1の25行目のSUM(合計)のような集合関数が指定されている. 集合関数には中間データが必要なので, 中間データは, 属性値が得られる実体に保持されるようにプログラムを生成するものとする. ところで, 中間データは何回も更新されるが, 入力データはソートされていないので, 同じ中間データが連続して更新されとは限らない. そこで, すべての中間データについて更新がすべて終了した後, その非主キー属性値を参照できるものとする. このため, その枝上の流れは, 非主キー属性節点を介してつながった他の枝の上の流れと同期しない. たとえば, 図3では枝A3が関連数Mを持つので非同期型になる.

c) 実体存在従属性制約によって実体が得られる有向枝の関連数が2以上かMの場合, その制約から同じ主キー値を持った実体が何個も得られるので,(1)項と同じように主キー値を照合しなければならない. ところが, 入力データはソートされていないので, 同じ主キー値を持った実体が連続して得られるとは限らない. そこで, すべての実体が得られた後, 実体の主キー属性値を参照できるものとする. このため, その枝の上の流れは, 実体型節点を介してつながった他の枝の上の流れと同期しない.

4.3 大局的な同期性の解析

グラフは前述の局所的解析の結果に基づいて大局的に解析する.

(1) 閉路による矛盾の検出と解決

図4(1)に示すような方向混在閉路の上で, 同図(2)に示すように非同期型有向枝がすべて同じ方向を持っていれば, 制約の実行順序に矛盾が起きる. その理由は, 非同期型有向枝を持った実体型節点か, 非主キー属性節点, 関連型節点があれば, その節点にすべての対象が流れ込んだ後でないと制約の実行は先へ進めないためである. この制約実行順序の矛盾を解消するには, 図4(3)に示すように反対方向の枝を少なくとも1本は非同期型に変えればよい.

(2) プログラムの大局的な最適化

前述の非同期型へ変える枝の選択は, 生成されるプログラムの大局的最適化の問題である. その解は, 局所的解析で得られた非同期型枝をすべて包含し, しかも(1)項の解析で矛盾を生じない非同期型枝の集合のうち, プログラムの効率が最もよいものとして得られる. しかし, この解法自体が大きな課題であるので, 後述の実験では以下の解法を用いた. まず, すべての方向混在閉路を解析して非同期型へ変わる枝の候補を検出する. その内, 最も多くの閉路の候補になっている枝から順に非同期型へ変える. その結果, 図3では枝A5が非同期型なので, 枝A6と, A7, A8, A9とが非同期型に変わった.

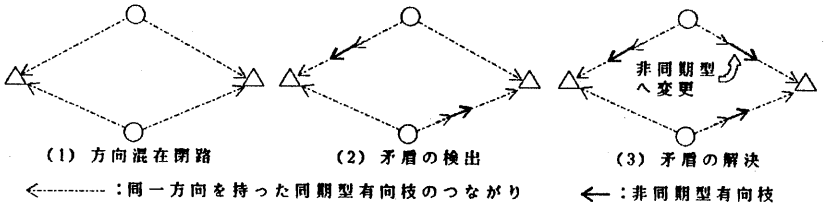


図4 閉路による矛盾の検出と解決

5 構造不一致の解決法

構造不一致は以下のようにプログラムのデータ構造と手続き構造を設計すれば解決できる.

5.1 データ構造の設計

データ構造は以下の順序で設計する.

(1) 構造節点の同期性

グラフの中で非同期型枝を持っている実体型節点と,非主キー属性節点,関連型節点とを非同期型とする.データセット型節点は常に非同期型である.その他の構造節点は同期型とする.

(2) 配列変数とスカラ変数の割当

非同期型の実体型節点と,非主キー属性節点,関連型節点には配列変数を割り当て,同期型の構造節点にはスカラ変数を割り当てる.配列の要素数はEN文とRN文に基づいて決める.たとえば,図3では節点productと,product.price, sale, sale.amount, buy, customer, customer.totalへ配列変数を割り当てられ,スカラ変数は節点sale.quantityとsoldとに割り当てられるので,データ構造は図5(1)のようになった.

5.2 手続き構造の設計

手続き構造は以下の順序で設計する.

(1) 有向グラフの分割

グラフを,以下のような同期型連結部分グラフに分割する.まず,非同期型節点を部分グラフの境界点に置き,同期型節点は部分グラフの内部に置く.節点を介してつながっている同期型枝は相互に同期しているので,同じ部分グラフの中に置く.制約節点を介して同期型枝につながっている非同期型枝も,その同期型枝と同じ部分グラフに入れる.部分グラフの中では対象の流れがすべて同期する.たとえば,図3のグラフは3つの部分グラフS1と,S2, S3とに分割された.

(2) 手続きブロックの割当と実行順序

各々の部分グラフへ手続きブロックを割り当てる.ところで大局的解析によって,隣接した部分グラフの境界上の非同期型節点は,その流入枝を一方の部分グラフの中に持ち,他方の部分グラフは流出枝のみを持つことが保証される.これは,隣接した部分グラフへ割り当てられたブロックの間に実行順序があることを示している.このため,ブロックの間に半順序ができるので,この半順序からブロック実行のための全順序を得る.

(3) 手続きブロック内の手続きの割当と実行順序

各々の部分グラフの中で,各制約節点と各構造節点へ手続きを割り当てる.これらの節点は有向枝で結ばれているので,手続きの間に半順序がある.この半順序から手続き実行のための全順序を得る.

(4) 手続きブロック内の繰り返しループの割当

各々の手続きブロックへ1つ以上の手続き繰り返しループを割り当てる.そのループの制御要因は以下のように部分グラフの中で最初に実行される制約で決まる.

a) 入力データのデータ層とアクセス層の結合制約が最初に実行される場合,入力ファイルへ1つのループを割り当て,入力レコード毎に実行を繰り返す.

b) 属性値従属性制約が最初に実行される場合,その制約に1つだけ記述されている関連型へ1つのループを割り当て,その関連型に属する関連毎に実行を繰り返す.

c) 関連存在従属性制約が最初に実行される場合,その関連型で対応づけられた実体型毎に1つづつのループを割り当て,各々のループは,実体型に属する実体毎に実行を繰り返す.

d) 実体存在従属性制約が最初に実行される場合,その制約から属性値を参照される非同期型の実体型節点が1つ以上あるので,その実体型毎に1つづつのループを割り当て,各々のループは,実体型に属する実体毎に実行を繰り返す.

e) 出力データのデータ層とアクセス層の対応制約が最初に実行される場合,その出力データのIX文のON句に1つだけ記述された実体型へ1つのループを割り当て,実体型に属する実体毎に実行を繰り返す.

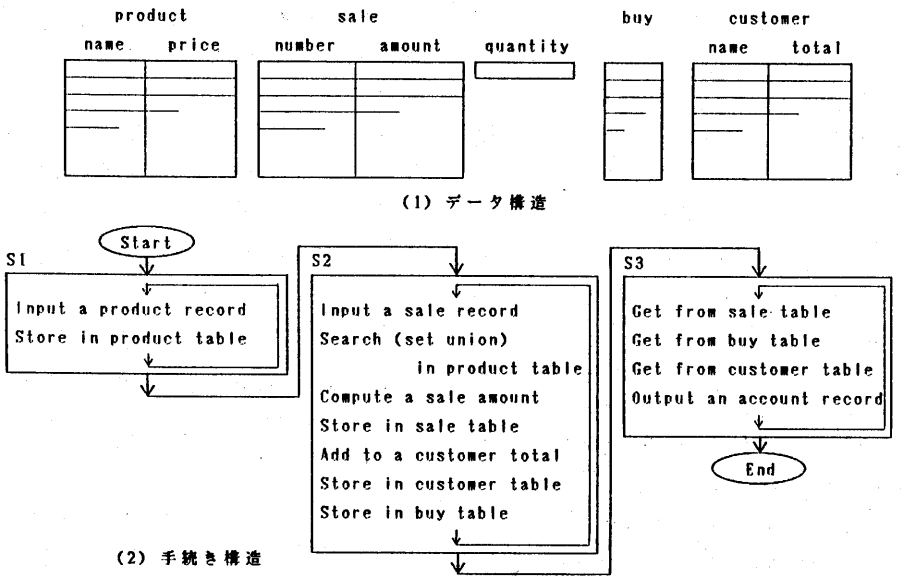


図5 プログラム構造

たとえば, 図3から図5(2)の手続き構造が得られた.

6 実験

前述の検出・解決法の妥当性を確認するため,PSDLプログラム仕様からCプログラムを生成する実験用コンパイラをSUNマシン上に作成した. 次章で述べるように検出・解決法にはまだ課題も多いので, 今後の拡張に備えてコンパイラは自己記述法によって作成した. 作成規模は約16000行であり, 一部はC言語で作成した. スケルトンは約500行であった. なお, 属性に割り当てた変数のデータ形式は一律に数値の場合は単精度浮動小数点数とし, 文字列の場合は64文字とした.

生成されたCプログラムの効率については,PSDL記述が100行以下のテスト・プログラム仕様を対象にして, 手書きのCプログラムと静的行数を比較した. その結果, 手書きの3倍の行数になった. なお, 比較はアセンブラ・プログラムに変換して行った.

7 考察

前述の検出法と解決法, 実験結果について考察する.

(1) 非手続き型言語の集合と写像の性質

第4章の構造不一致検出では和集合演算の必要性や写像の数量関係を解析した. また, 第5章の構造不一致解決では集合の性質を持つ実体型や関連型へテーブルを割り当てるとともに, 写像による連結性を解析してグラフを分割した. すなわち, 本稿の検出・解決法は,MODEL言語の配列とはまったく異なった集合と写像の性質を用いた. 集合と写像は, 実体型や関連型のように概念を直接表すのに適しているので, 高位の非手続き型言語に採用される傾向がある. それらのコンパイラを作成するのに本稿の検出・解決法は参考になるものと予想される.

(2) 構造不一致の検出と解決の精度向上

本稿では次の点を考案することによって, 文献[6]よりも構造不一致の検出と解決の精度が向上した.

a) 同期性の性質は文献[6]では有向グラフの節点のみへ持たせたが, 本稿では有向枝へも持たせた. このため, 大局的解析では非同期型節点で交差した有向枝が同期型と非同期型に分かれることができ, グラフ分割では非同期型節点を介した2本の同期型枝が同じ部分連結グラフへ入ることが可能となった.

b) 文献[6]では同じ節点から分流して他の節点へ合流する有向道群を用いて大局的解析を行った. その解析方法を, 本稿では方向混在閉路による解析へ一般化した.

c) その他に本稿では関連型節点や関連存在従属性制約節点も導入した. また, 非同期型節点となった実体型の属性が同期型節点になるのも可能となった.

今後の課題として, さらに検出と解決の精度を向上させるには, 隣接した有向枝間の関係として同期性の性質

を取り扱うことなどが考えられる.

(3) 大局的な最適化

本稿では4.3節で述べたように一種の大局的最適化の問題を具体的できた. しかし, 最適化の解法自体が課題であるので, 今後は種々の最適化アルゴリズムの適用も含めて解法を研究することが必要である.

(4) 生成されたプログラムの効率

PSDLのコンパイラから生成されたプログラムの効率は第6章に述べたとうりである. 実験用に作成された本コンパイラのスケルトンはさらにきめ細かく洗練できるので, 効率は改善する余地がある. このため, 本章に述べた課題を解決していけば, 実用的な効率を備えたプログラムを生成できるものと予想される.

(5) 構造不一致と検出・解決法の定式化

第3章の(3)項で述べた構造不一致の統一的な見方を定式化して構造不一致を分類するとともに, 検出・解決法, 特に本稿では未解決の単一方向閉路の問題や, 順序不一致, 境界不一致等の検出・解決法を定式化することが必要である.

8 おわりに

非手続き型言語の集合と写像の性質に着目した入出力データの構造不一致検出・解決法とその実験について述べた. 集合と写像は高位な非手続き型言語に採用される傾向があるので, それらのコンパイラを作成するのに本稿の構造不一致検出・解決法は参考になるものと予想される.

構造不一致を検出して解決するのに, プログラム仕様から作成された有向グラフの枝へ同期性の性質を持たせた. その上で方向混在閉路の解析や, グラフの部分連結グラフ分割を行うことなどによって, 検出と解決の精度が向上した.

この検出・解決法に基づいて実験用のコンパイラを作成して評価したところ, 生成されたプログラムの静的行数は手書きのものと比較して約3倍であった. さらに, 生成されたプログラムの大局的最適化の問題を具体的できた. このため, 後述の課題を解決していけば, 実用的な効率を備えたプログラムを生成できるものと予想される.

今後の課題としては構造不一致を定式化して構造不一致の分類を行い, その分類に従って本稿では未解明の単一方向閉路の問題や, 順序不一致, 境界不一致などの検出・解決法を定式化する. また, 検出精度の向上や, 生成されたプログラムの大局的最適化についても研究の予定である.

謝辞 日頃ご指導いただく葉原会長, 山下社長, 竹中室長に深謝いたします. また, ご討論いただいた研究室の諸氏, ならびにPSDLコンパイラの作成にご協力いただいた日本電子計算株式会社の諸氏に感謝いたします.

参考文献

[1] M.A. Jackson (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] N.S. Prywes, A. Pnueli: Compilation of Nonprocedural Specifications into Computer Programs, *IEEE Trans. Software Eng.*, Vol. SE-9, No. 3, pp. 267-279 (1983).

[3] 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).

[4] 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).

[5] K. Okamoto, M. hashimoto: On Real-Time Software Specification Description with a conceptual Data Model-Based Language, *Proc. of ICCI 90* 掲載予定.

[6] 橋本正明: 非手続き型言語と入出力データの構造不一致, 情報処理学会論文誌, Vol. 29, No. 12, pp. 1141-1150 (1988).

[7] 橋本正明: プログラム構造設計の自動化について, 情報処理学会CASE環境シンポジウム論文集, pp. 101ー 108 (1989).

[8] M. hashimoto, K. Okamoto: A Set and Mapping-based Detection and Solution Method for Structure Clash between Program Input and Output Data, *Proc. of COMPSAC 90* 掲載予定.

[9] C.M. Hoffmann: Design and Correctness of a Compiler for a Non-Procedural Language, *Acta Informatica*, Vol. 9, pp. 217-241 (1978).