入出力データの構造不一致検出における処理ネック解決と精度向上

橋本正明† 廣田豊彦† 横田和久††

ソフトウェアの生産性向上や信頼性向上は大きな課題である．著者らはその解決策として，理解性と拡張性にすぐれた仕様記述言語PSDLと，PSDLで記述された仕様からプログラム構造を自動設計する手法を既に提案した．仕様からプログラム構造を設計する際の課題の一つが，入出力データの構造不一致の検出と解決である．著者らは，仕様から有向グラフを生成し，そのグラフ中の閉路上でデータの流れの同期性を解析することによって，構造不一致を検出し，解決する手法を既に提案した．しかし，この手法では，仕様が大きくなるに従って，グラフの閉路数が指数関数的に増大し，処理ネックを生じていた．また，データ検索などの処理において，構造が一致しているにもかかわらず，構造不一致と判定される場合があった．本論文では，閉路の代わりにカットセットを対象として，すべての解を探索しなくてもすむ解析手法を新たに提案し，実験によって，その手法が最適性の面でそれほど劣らないことを確認した．また，グラフの隣接する二つの枝を調べることによって，構造不一致の検出精度を改善する手法を考案した．以上の二つの手法を取り入れてプログラム生成の実験を行い，正しく動作するプログラムが生成されることを確認した．

A Performance Bottleneck Solution and Accuracy Improvement in the Detection of Structure Clash between Input and Output Data

MASAAKI HASHIMOTO,† TOYOHIKO HIROTA † and KAZUHISA YOKOTA ††

Software productivity and reliability are the main concerns in software engineering. To improve them, the authors have already proposed specification description language PSDL with high comprehensibility and extensibility, and a method to automatically design a program structure from specifications written in PSDL. One of the difficulties in designing a program structure from specifications is to detect and solve structure clash between input and output data. The authors have already proposed a detection and solution method which generates a directed graph from specifications and analyzes synchronousness of data flow in loops of the graph. However, this method has a performance bottleneck because the number of loops increases exponentially as specifications grow in size. Another problem is that the method may detect structure clash even if there is no clash when the specification includes retrieval processing. This paper proposes a new method which analyzes cutsets instead of loops and does not search all the cases. An experiment has proved that this method is not inferior in optimality. Moreover, this paper proposes a method to improve the detection accuracy of structure clash by analyzing two adjacent arcs in a graph. The authors has experimented on program generation to which the above two improvements has been applied, and been assured that the generated programs work correctly.

1. はじめに

ソフトウェアの生産性向上や信頼性向上は大きな課題であり，その一方策としてソフトウェア再利用が着目されている[1]．実際，COBOLのような手続き型プログラムの再利用は，生産性向上や信頼性向上に寄与してきた[2]．最近はソフトウェア開発に現れる知識の再利用，すなわち領域知識や開発経験，仕様などの再利用が研究されている[3]．著者らも知識再利用研究の一環として，仕様の再利用を研究中である[4]．ところで，仕様を再利用するにはその仕様を理解のうえ，新しい要求に合わせて拡張するので，再利用対象仕様は理解性と拡張性を備えていなければならない．そこで著者

† 九州工業大学情報工学部知能情報工学科
Department of Artificial Intelligence, Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology
†† 横河・ヒューレット・パッカード株式会社アジアパシフィックプロダクト開発本部システム環境開発部
Asian System Laboratory, Asia Pacific Product Operation, Yokogawa-Hewlett-Packard, Ltd.

らは，理解性の高い仕様の概念モデルとして，ERモデル（entity relationship model）[5]を採用し，計算を非手続き的に記述することによって拡張性を高めた仕様記述言語PSDL（Program Specification Description Language）[6),7)]をすでに提案した．

このような言語は計算機で直接実行できないので，著者らはプログラム仕様から手続き型プログラムの構造を自動設計する方式を考案し，Cプログラムや C++プログラムを生成するコンパイラを研究中である[6),8)]．ところで，実行効率のよいプログラム構造を設計するには，Jacksonの構造化プログラミング法[9)]の主題である入出力データの構造不一致を検出し解決しなければならない．しかし，構造不一致の自動的な検出解決法の研究は少なく[10)]，特にERモデルの集合と写像の性質に着目した研究はなかった．

そのため，著者らは集合と写像の性質に着目した構造不一致検出解決法をすでに提案した[6)]．しかし，構造不一致検出の際，プログラム仕様から得られた有向グラフの中ですべての閉路について，データフローの同期性に関する，デッドロックに似た矛盾を解決しているので，指数関数的に増加する閉路数[11)]によって処理ネックが生じていた．そこで閉路の代わりにカットセットに基づいて，構造不一致検出の性能を向上させる改善策を考案した．また，従来はグラフの個別の枝を対象にして，データフローの同期性を解析していたが，隣接する二つの枝に着目する解析を導入することによって，構造不一致の検出精度を改善した．

本論文の目的は，上記のカットセットに基づく解析法と，二つの枝間の関係を対象にした解析法とを提案し，それらの実験を報告することである．2章に本研究の背景を述べ，3章でPSDLの概要とその有向グラフ表現について述べる．4章では局所的構造不一致と大局的構造不一致について述べる．5章で大局的構造不一致を解決するためのカットセット解析法について述べ，6章で構造不一致検出精度の改善を述べる．7章では実験を報告し，考察を行う．

2. 研究の背景

（1） 入出力データの構造不一致

製品名とその単価から成るレコードを格納した製品ファイルと，売上ごとの売上番号と売上製品名，売先顧客名，売上数量から成るレコードを格納した当月売上ファイルとを順アクセスで入力し，売上ごとに売上額とその売先顧客の当月売上合計額から成るレコードを格納した計上ファイルを順アクセスで出力する販売管理プログラムを作成する場合，そのプログラム構造は以下のように設計される．

売上額を計算するには売上数量と単価が必要であるが，製品ファイルと売上ファイルを製品名でソートしていなければ，同じ製品名を持つ製品レコードと売上レコードは同期して入力することはできない．また，売上レコードをすべて入力しなければ顧客の当月売上合計額は得られないので，売上レコードの入力と，顧客の当月売上合計額を含んでいる計上レコードの出力とは同期しない．この非同期問題は，Jacksonの構造化プログラミング法[9)]で**入出力データの構造不一致**と呼ばれている．そこで構造不一致を検出すると，製品ファイルのデータや，売上数量を除く売上ファイルのデータ，顧客ごとの当月売上合計額を，プログラム内のテーブル，または中間ファイルに一時保持して，同期がとれるように構造不一致を解決したプログラム構造を設計する．

（2） 自動的な構造不一致検出解決法

Prywesらは配列変数上で計算式を定義する非手続き型言語MODEL[10)]を提案し，その言語で記述されたプログラム仕様から自動的に構造不一致を検出し解決してプログラム構造を設計する方式を提案した．この方式は配列の指標の性質に着目して，構造不一致を検出し解決している．一方著者らは，最近よく利用されているERモデルの上で従属性制約を定義する非手続き型言語PSDLを提案し，エンティティ型や関連型の集合としての性質，および関連型の写像としての性質に着目して，自動的な構造不一致検出解決法とプログラム構造設計法を提案した．

（3） 構造不一致検出法の問題点

著者らの構造不一致検出法では，プログラム仕様から得られた有向グラフの上で，データフローの同期性を解析する．その際，有向グラフ中の全閉路上で，データフローの同期性に関する，デッドロックに似た矛盾を解決しなければならない．しかし，閉路数は最悪の場合，指数関数的に増加する[11)]ので，全閉路を直接解析する検出法[6)]には処理ネックが存在していた．このような問題には，領域知識に基づいて計算量を減らすヒューリスティックな解析法が有効とされている．

また，従来はグラフの個別の枝を対象にして，データフローの同期性を解析していた．しかし，この解析法では構造不一致の検出精度が十分ではなく，データの検索に類した処理は実際に構造が一致していても，構造不一致として検出されていた．

3. PSDLと有向グラフ

3.1 PSDL

販売管理プログラムの仕様をPSDL[6]で記述した例を**図1**に示す．PSDLで記述された仕様は情報層，データ層，アクセス層から構成され，それぞれ，INFORMATION文，DATA文，ACCESS文で表される．

情報層では，対象世界の情報構造を**ERモデル**で記述する．E文がエンティティ型を，R文が関連型を表す．また，エンティティの属性値を計算するための**従属性制約**を＝で記述する．図1の例では，product, sale, customerの三つのエンティティ型と，sold, buyの二つの関連型がある．そしてsaleの属性amountと，customerの属性totalに従属性制約が記述されている．

データ層では，基本，連接，繰り返し，選択の4種のデータ型の階層によって，入出力データのデータ構造を記述する．この層のデータと情報層との関係は**情報制約**として，＝で記述する．図1の例では，product_data, sale_data, account_dataの三つのデータ構造があり，それぞれのフィールドに対して情報制約が記述されている．

アクセス層では，入出力ファイルへのアクセス方法をデータセット型で記述する．図1の例では，二つの入力ファイルproduct_file, sale_fileと一つの出力ファイルaccount_fileが用いられる．

3.2 有向グラフへの変換

与えられたPSDLプログラム仕様から，以下のような節点を生成する．

1. エンティティ型および関連型に対して，それぞれ節点を生成する．
2. エンティティの属性（A文で記述）に対しても，それぞれエンティティ型とは別に節点を生成する．ただし，主キー属性（K文で記述）はエンティティ型節点に対応させるので，別の節点は生成しない．
3. データ型およびデータセット型に対して，それぞれ節点を生成する．ただし，データ型については，最上位レベルのデータ型でそれ以下のデータ型を代表させるものとし，階層全体で一つの節点とする．
4. 情報層，データ層の各種制約に対して，それぞれ節点を生成する．またアクセス層の記述（データ型とデータセット型の関係）に対してもそれぞれ節点を生成する．

これらの節点のうち，1～3を**構造節点**とよび，4を**制約節点**とよぶ．

つぎに，それぞれの制約記述において参照されている構造節点とその制約節点との間に枝をおく．またアクセス層の記述に対しても対応する構造節点と制約節

```
INFORMATION
  E product
      EN -50
    A name  STR
     K
    A price NUM
  R sold
    C .product
      RN M
    C .sale
      RN 1
  E sale
      EN -100
    A number   NUM
     K
    A quantity NUM
    A amount   NUM
      = quantity * .sold..product.price
  R buy
    C .customer
      RN M
    C .sale
      RN 1
  E customer
      EN -50
    A name  STR
     K
    A total NUM
      = ASUM(.buy..sale.amount)
DATA
  I product_data
       IX product_id
    G product_record ON EndOfFile(product_data)
    O product_record
       %12s product_name
         = product.name
       %8d  product_price
         = product.price
  I sale_data
       IX sale_id
    G sale_record ON EndOfFile(sale_data)
    O sale_record
       %4d  sale_number
         = sold..sale.number
         = buy..sale.number
       %16s sale_customer
         = buy..customer.name
       %12s sale_product
         = sold..product.name
       %4d  sale_quantity
         = sale.quantity
  I account_data
       IX account_id
    G account_record ON PEntityNumber(sale)
    O account_record
       %4d  sale_number
         = sale.number
         = buy..sale.number
       %8d sale_amount
         = sale.amount
       %16s sale_customer
         = buy..customer.name
       %10d customer_total
         = buy..customer.total
ACCESS
  D product_file INPUT  20 product_data
  D sale_file    INPUT  36 sale_data
  D account_file OUTPUT 38 account_data
```

図1 PSDLプログラム仕様例
Fig. 1 Example of PSDL program specifications.

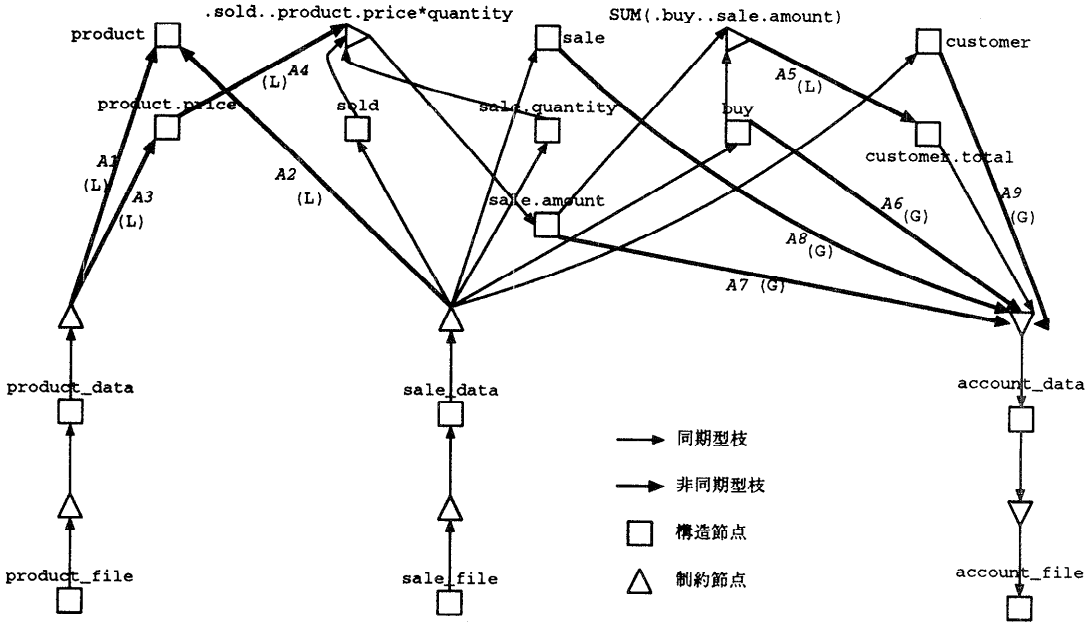


図2 有向グラフの例
Fig. 2 Example of directed graph.

点の間に枝をおく．枝の方向は，制約で記述されるデータフローの方向に従って決める．PSDL ではデータフローは必ず一意に決まらなければならないので，各枝の方向は一意に決まる．なお，同一方向の枝だけからなる閉路は存在しないものとする．

図1に示した PSDL プログラム仕様に対する有向グラフを図2に示す．

4. 構造不一致

PSDL プログラム仕様における構造不一致は，局所的構造不一致と大局的構造不一致に分けられる．照合，一対多関係，集合演算など，データの集合を扱う演算が必要な場合に生じる構造不一致を局所的構造不一致とよぶ．一方，演算自体は単一のデータが対象であっても，システム全体のデータフローがデッドロックを起こす場合があり，これを大局的構造不一致とよぶ．

4.1 局所的構造不一致

局所的構造不一致が生じるのは，つぎの二つの場合である．

1. エンティティ型に対応する節点に二つ以上の枝が流入する．
2. 制約が参照する関連型において1対多または多対多の関連があり，RN 文で多重度 M が指定されている．

1の場合は，二つ以上の枝を通じて流入するエンティティの照合を意味する．図2の例では，エンティティ型 product の節点へ，データ型 product_data とデータ型 sale_data から，それぞれ制約を通じて枝 $A1$ と $A2$ が流入している．この場合，product_data からのエンティティは主キー name に関してソートされているかもしれないが，sale_data が product_name に関してソートされていることは一般には保証されない．したがって，この両者のデータを照合しようとすると構造不一致が起こることになる．このとき，図2の枝 $A3$ のように，主キー属性以外の属性に関しても構造不一致が起こる．

2の場合は，制約に流入する枝と制約から流出する枝に分けられる．流入する枝に対応する関連型の多重度が M であることは，その枝の始点が2度以上参照されることを意味している．図2では枝 $A4$ がこれに相当する．一方，流出する枝に対応する関連型の多重度が M であることは，その制約が集合関数であることを意味している．図2では枝 $A5$ がこれに相当する．いずれの場合でも，個々のデータに同期して処理を行うことはできず，構造不一致となる．

以上のような構造不一致を解決するには，データの流れを非同期化すればよい．すなわち，構造不一致が生じる枝のところでデータを蓄え，すべてのデータがそろった後に次の処理を始める．これを有向グラフ上で表示するために，枝に同期型/非同期型のラベルを付けるものとする．図2で（L）が記されている枝 $A1$, $A2$, $A3$, $A4$, $A5$ が非同期型であり，その他の枝は，局

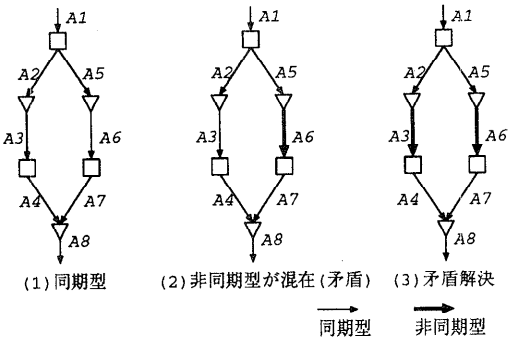


図3 大局的構造不一致の例(1)
Fig. 3 Example of global structure clash (1).

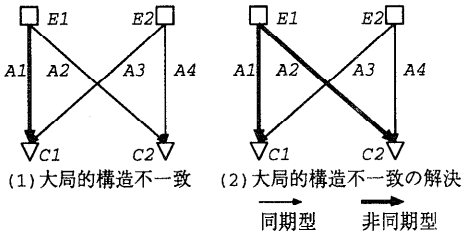


図4 大局的構造不一致の例(2)
Fig. 4 Example of global structure clash (2).

所的構造不一致に関しては同期型である.

4.2 大局的構造不一致

局所的構造不一致が解決されても，システム全体のデータフローがデッドロックあるいは矛盾を起こすことがある.

簡単な例として**図3**(1)に示すような有向グラフを考えてみる．すべての枝が同期型であれば，データフローは問題なく流れる．ところが，図3(2)に示すような非同期型枝 $A6$ があると，すべてのデータが到着するまで，次の枝 $A7$ へはデータは流れない．一方，枝 $A2, A3, A4$ には枝 $A5$ と同期してデータが流れる．このため枝 $A4$ と $A7$ の合流点において矛盾が生じることになる．この矛盾を解消するには，図3(3)に示すように，枝 $A2, A3, A4$ のいずれかを非同期型に変更すればよい.

二つのエンティティ型による矛盾も生じることがある．**図4**(1)にその例を示す．枝 $A1$ が非同期型であるとすると，エンティティ型 $E1$ のデータをすべて処理しないと制約 $C1$ を計算することはできない．一方枝 $A2$ は同期型であるので，制約 $C2$ はエンティティ型 $E1$ のデータフローと同期して計算されなければならず，エンティティ型 $E2$ のデータフローも同期しなければならないことになる．すると枝 $A3$ にエンティティ型 $E2$ が同期して流れることになり，枝 $A1$ が非同期型であることに矛盾する．この矛盾を解消するには，図4(2)に示すように，枝 $A2$ または $A3$ のいずれか一方を非同期型に変更すればよい.

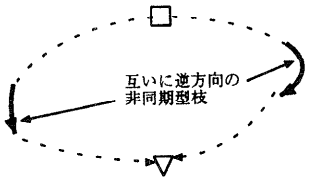


図5 大局的構造不一致の解決
Fig. 5 Solution of global structure clash.

以上の場合を一般化すると，任意の閉路中に非同期型枝が存在するとき，その逆方向の非同期型枝が存在しなければ構造不一致が起こることがわかる(**図5**)．これは閉路が四つ以上の有向道* から構成される場合でも成立する.

5. 大局的構造不一致の解決

大局的構造不一致を検出して解決するには，4.2節で述べたように，非同期型枝を含むすべての閉路を調べなければならない．しかし，グラフ中の閉路数は最悪の場合，枝数の指数で増加する[11]．このため，大局的構造不一致の検出・解決を閉路解析によって行うことは現実的ではないと考えられる．実際，閉路解析法による従来の PSDL コンパイラ[6]で 176 行のプログラム(有向枝数 149，節点数 73) をコンパイルしたところ，SUN 3/80 上で 24 時間経過しても終了しなかった.

そこで閉路の代わりにカットセットについて考えてみる．閉路とカットセットは双対の関係にあり，すべてのカットセットを解析することとすべての閉路を解析することはほぼ同じだけの計算量を要する．しかし，大局的構造不一致の解析においては，以下で述べるカットセットの性質を利用することができる.

対象となる有向グラフは一方向閉路を持たないので，節点の間には半順序が成立する．すなわち，枝の始点 N_i と終点 N_t に対して，$N_i \leq N_t$ と定義することができる．このとき，ある一つの枝 A に着目し，その始点と終点をそれぞれ含み，半順序関係に矛盾しないように全節点集合を上下二つの集合に分割することができる(**図6**)．このとき，カットセットにその枝 A が含まれ，カットセット中のすべての枝は，下位集合から上位集合へ向いている.

また，枝 A の始点と終点がそれぞれ別の集合に含ま

* 同一方向の枝からなる経路．同一方向の有向道が隣接するときには一つに併合する．したがって，閉路中の有向道の数は偶数になる.

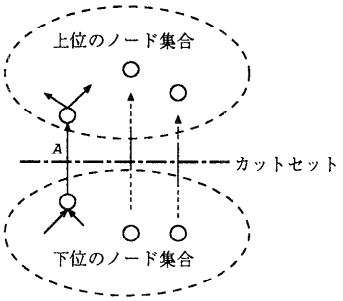


図6 同一方向の枝からなるカットセット
Fig. 6 Cutset of arcs with the same direction.

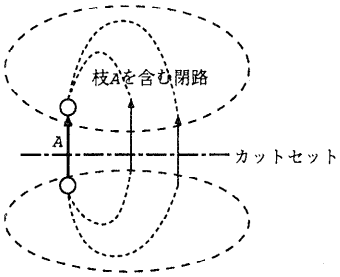


図7 カットセットと閉路
Fig. 7 Cutset and loop.

れるので，枝 A を含む閉路は，A 以外の枝で同じカットセットに含まれるような枝が存在する（図7）．以上を整理すると以下のようになる．

カットセットの性質

1. 一方向閉路を持たない有向グラフにおいては，任意の枝に対して，その枝を含み，しかも同一方向の枝だけから構成されるようなカットセットが存在する．
2. ある枝 A を含む任意のカットセットを選んだとき，枝 A を含むどんな閉路でも，閉路中の枝 A 以外の枝が一つ以上必ずそのカットセットに含まれている．

これらの性質に基づいて，局所的構造不一致によって非同期型に変更されたすべての枝に対して，以下の手順を適用することによって，大局的構造不一致を解決することができる．

大局的構造不一致の解決手順

1. 対象となる枝を含む同一方向の枝からなるカットセットを一つ選び出す．
2. そのカットセットに含まれるすべての枝を非同期型に変える．

カットセットの性質1から手順1が保証される．また，図7からわかるように，手順2を実行すると，どの閉路も逆方向の非同期型枝（図5参照）を持つことになり，大局的構造不一致が解消される．

表1 非同期型枝の数の比較
Table 1 Comparison of the number of asynchronous arcs.

プログラム	枝数	カットセット解析後の非同期型枝数			
		$n=1$	$n-2$	$n=3$	$n=4$
BGA	26	6	6	6	6
AV	31	9	9	9	9
GIC	167	61	61	61	59
GAI	180	85	86	85	85
GII	199	90	89	90	90

実測時間(SUN 3/80)：1 sec($n=1$〜2)，1〜10 min($n=4$)．

カットセットは性質1を満たすものであれば，どのようなものでもよいので，すべてのカットセットを調べる必要はない．しかし，仕様から生成されるプログラムの効率を考えると，処理時間が極端に増大しない範囲で，可能なかぎり非同期型枝の数が少なくなるようにカットセットを選ぶべきである．そこで本研究では，ある n ($n\geq 1$) を定め，以下のような手順を用いることにした．n の値については後で議論する．

カットセットの選択アルゴリズム

1. 解析対象の非同期型枝に着目し，すべての節点を半順序に基づいて，その枝の始点よりも上位の節点集合 N_a，終点よりも下位の節点集合 N_b，いずれでもない節点集合 N_c に分類する．
2. グラフを N_a と $N_b \cup N_c$ に分割する．
3. N_c 中の n 個以下の節点を含むすべての部分集合を生成する．ただし，その部分集合中の節点を N_c から N_a に移したときに，半順序に従って必然的に N_c から N_a に移る節点があるときには，その節点も部分集合に加える（結果的に集合の要素数が n を越えることもある）．
4. 生成された各部分集合に対して，その集合中の全節点を N_c から N_a へ移した場合のカットセット中の枝数の増分値 h を計算する．

$$h=\text{流出枝数}-\text{流入枝数} \qquad (1)$$

5. h が負の値で，最小であるような部分集合に含まれる節点を，N_c から N_a へ移す．h が負となる部分集合が存在しないときには，アルゴリズムを終了する．さもなければ2へ戻る．

n の値を N_c の要素数とすると，すべてのカットセットを調べることになり，すでに述べたように現実的ではない．そこでいくつかの例題に対して，$n=1$〜4 としたときに抽出される非同期型枝の数を調べてみた．その結果が**表1**である．それぞれの例題ごとに，$n=1$〜4 の非同期型枝数を比較すると，その差は高々2で

あり，非同期型枝の総数の5%以下である．この結果から，$n=1$としてもそれほど最適性が劣るわけではないと判断でき，本研究では$n=1$とした．

図2の例で，非同期型枝$A5$に対して上記のアルゴリズムを適用すると，accountに関係する節点とcustomer.totalがN_aに，customer, sale, productがN_cに，その他のすべての節点がN_bに含まれることになる．N_cに含まれる3個の節点のうち，productは$h=2$, saleとcustomerはいずれも$h=0$となるので，節点の移動は起こらず，$A5, A6, A7, A8, A9$からなるカットセットが選ばれる，したがって，(G)が付いている枝$A6, A7, A8, A9$が新たに非同期型になる．

6. 構造不一致検出精度の改善

前章までで述べた構造不一致検出法では，構造が一致しているにもかかわらず，構造不一致として検出されるものがある．たとえば，図2でsale.amount値を決定する制約が，図8(1)に示すように，product.nameも参照すると仮定する．そうすると，その制約節点へ構造節点productから流入する枝Aaができ，その枝を含む閉路ができる．このため，枝$A2$の非同期性が大局的解析で波及して，構造節点soldにつながる枝AbまたはAcが非同期型になる．しかし，$A2$は前述の制約と同期してエンティティproductを検索するので，枝Ab, Acとも本来同期型でよい．

このような過剰な非同期性が生じる要因は次の二つである．

過剰な非同期性の要因

1. 一つの枝を流れるデータが二つの目的で用いられる．
2. 二つのうちの一方を参照する制約が必ず同期的であることが解析に反映されていない．

この問題を解決するためには，隣接する二つの枝の組合せの同期性を調べる必要がある．これを簡単に行うために，以下のような手順で有向グラフを変換する．

有向グラフの変換手順

1. 局所的解析の後，構造節点を介して隣接する，同一方向を持つ2本の枝の組を各々1本の枝に変え，構造節点を除去する．元の2本の枝が両方とも同期型の場合，新たな枝も同期型とし，その他の場合は非同期型とする．
2. 1にかかわらず，以下のすべての条件を満たす場合には同期型とする．
   (a) 情報制約節点からエンティティを経由して制約節点に流入する枝である．
   (b) この枝の元の2本の枝のうち，エンティティから制約への枝は同期型である．
   (c) 同じ制約節点の組の間に，同一方向で関連を経由する枝が存在する．

上記の手順2の3条件は，先に述べた過剰な非同期性の要因をグラフ上で検出するためのものであり，その条件に合致する枝は同期型に戻している．図8(1)の例に対してこの変換を行うと，図8(2)のようになる．ここで枝$A2a$が手順2の条件を満たして同期型となる．

7. 実験と考察

従来のPSDLコンパイラに適用していた閉路解析法を，5章で述べた$n=1$のカットセット解析法で置き換え，さらに6章で述べた構造不一致検出精度の改

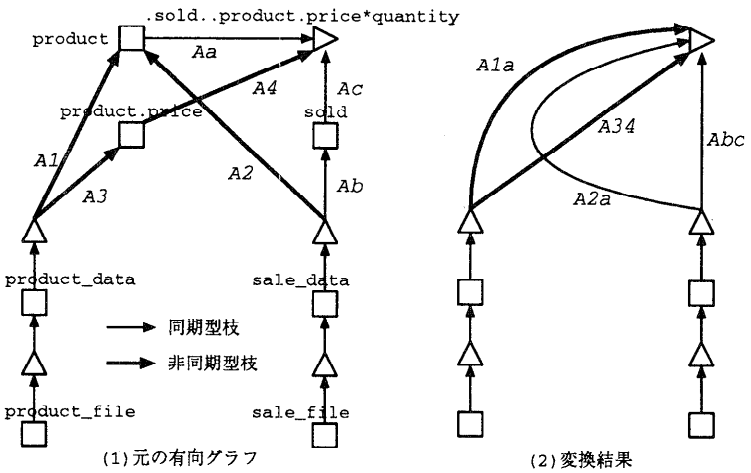


図8 有向グラフの変換
Fig. 8 Transformation of directed graph.

善を施し実験した．その結果，生成されたCプログラムは正しく動作した．また，新旧コンパイラの処理時間を比較する実験を行った．その実験結果を**表2**に示す．表2から分かるように，大局的解析の処理時間が大幅に改善されており，その結果としてコンパイル時間全体も改善された．現在，500行のPSDLプログラム仕様がコンパイルでき，その処理時間は3分であった（SUN 3/80のユーザ時間）．

以下，考察と今後の課題を述べる．

（1） 処理ネックの解決

カットセット解析法の考案によって大局的解析の処理時間が大幅に改善され，大きなプログラム仕様も許容時間内でコンパイルできることを確認し，処理ネックが解決した．この手法はすべての場合を尽くしているわけではないが，構造不一致検出精度がそれほど劣らないことも確認した．

PSDLコンパイラには部分グラフ分割のような他のグラフ処理もあるが，閉路解析のような致命的問題はない．新コンパイラには処理可能なプログラム仕様が500行という制限はあるが，この制限はコンパイラ自体をPSDLで記述したことに起因している．したがってコンパイラを，たとえばC言語で作成すれば，この制限は解決できる．

（2） 構造不一致検出精度の改善

同期性の解析対象を有向枝から，隣接有向枝間の関係へ変更した．これによって，エンティティの和集合演算と検索とを区別する解析法を導入することができ，データ検索を行う被生成プログラムの実行効率が改善した．さらに処理形態に応じて，さまざまな解析法を導入するのは今後の課題である．

（3） 被生成プログラムの実行効率最適化

被生成プログラムの実行効率最適化は主に大局的解析に依存する．しかし，上記のカットセット解析と閉路解析は非同期型枝を少なくするだけなので，実行効率を最適化しているわけではない．将来はプログラムの実行時間とメモリ量を考慮する最適化法を研究する必要がある．

表2 コンパイル時間の比較
Table 2 Comparison of compile time.

プログラム	旧コンパイラ			新コンパイラ		
	全処理時間 T	大局的解析 G	G/T	全処理時間 T	大局的解析 G	G/T
TP1	15.0	4.5	0.30	9.5	0.1	0.01
TP2	9.1	1.9	0.23	7.7	0.1	0.01
TP3	13.9	2.7	0.19	11.5	0.2	0.02
TP4	15.4	1.0	0.06	14.0	0.3	0.02

コンパイル時間(sec)：SUN 3/80で実測．

8. まとめ

PSDLプログラム仕様から得られる有向グラフにおいて，構造不一致を起こす非同期型枝を含むカットセットは，その枝を含むすべての閉路上でデータフローの矛盾を解決できる．そこで許容される時間内に好ましいカットセットを見つける解析法を述べた．さらに実験によって，閉路解析で生じていた処理ネックを解決し，しかも構造不一致の検出精度が劣らないことを確認した．また，同期性の解析対象を有向枝から，隣接枝間の関係へ変えたことで，データ検索を行う被生成プログラムの実行効率を改善することが可能になった．今後，領域知識に基づく解析法をさらに導入する必要があるが，これまでの有向グラフのような静的モデルでは限界があり，ペトリネットのようにデータの動的な流れのモデル化に関する研究が必要である．また，プログラムの実行時間とメモリ量を考慮する最適化法を研究することも重要である．

謝辞 本論文はATR通信システム研究所からの受託研究「ソフトウェアモデリングに関する研究」で完成したものであり，同研究所の諸氏に深謝いたします．また，SoftReuseシステムの試作と実験にご協力いただいた日本電子計算株式会社の諸氏に深謝いたします．

参考文献

1) Brooks, F. P.: No Silver Bullet: Essence and Accidents of Software Engineering, *IEEE Comput.*, Vol. 20, No. 4, pp. 10-19 (1987).
2) Lanergan, R. G. and Grasso, C. A.: Software Engineering with Reusable Designs and Code, *IEEE Trans. Softw. Eng.*, Vol. SE-10, No. 5, pp. 498-501 (1984).
3) Iscoe, N., Williams, G. B. and Arango, G.: Domain Modeling for Software Engineering, *Proc. 13th Int. Conf. Softw. Eng.*, pp. 340-343 (1991).
4) Okamoto, K. and Hashimoto, M.: Visual Programming with Reusable Specification Described by a Conceptual Data Model- and Constraint-Based Language, *Proc. Int. Conf. Human-Computer Interaction*, pp. 587 - 591 (1991).
5) Chen, P. P.: The Entity-Relationship Model—Toward a Unified View of Data, *ACM Trans. Database Syst.*, Vol. 1, No. 1, pp. 9-36 (1976).
6) Hashimoto, M. and Okamoto, K.: A Set and Mapping-Based Detection and Solution Method

for Structure Clash between Program Input and Output Data, *Proc. IEEE Computer Software and Application Conference*, pp. 629-638 (1990).
7) Okamoto, K. and Hashimoto, M.: On Real-Time Software Specification Description with a Conceptual Data Model-Based Language, *Proc. Int. Conf. Computing and Information*, pp. 186-190 (1990).
8) 山崎充彦，廣田豊彦，橋本正明：仕様記述言語PSDLからオブジェクト指向言語への変換，第48回情報処理学会全国大会論文集，1G-10 (1994).
9) Jackson, M. A.: *Principles of Program Design*, Academic Press (1975).
10) Prywes, N. S. and Pneuli, A.: Compilation of Nonprocedural Specifications into Computer Programs, *IEEE Trans. Softw. Eng.*, Vol. SE-9, No. 3, pp. 267-279 (1983).
11) Mateti, P. and Deo, N.: On Algorithms for Enumerating All Circuits of a Graph, *SIAM J. Comput.*, Vol. 5, No. 1, pp. 90-99 (1976).

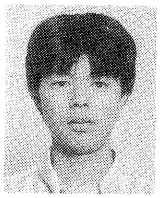
(平成6年8月31日受付)
(平成7年2月10日採録)

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．平成5年九州工業大学情報工学部教授．従来オペレーティングシステムや，データベース設計，仕様記述言語，ソフトウェア自動作成等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，IEEE等各会員．

廣田　豊彦（正会員）

1954年生．1976年京都大学工学部電気工学第二学科卒業．1981年同大学大学院工学研究科博士後期課程研究指導認定退学．工学博士．1981年京都大学情報処理教育センター助手．1988年九州工業大学情報科学センター助教授．1989年同大学情報工学部知能情報工学科助教授，現在に至る．プログラム開発環境，プログラムテスト支援，CADシステムなどの研究に従事．著書「Cプログラミングの基礎」(培風館)ほか．日本ソフトウェア科学会，人工知能学会各会員．



横田　和久（正会員）

1964年生．1988年電気通信大学電気通信学部計算機科学科卒業．1988年横河・ヒューレット・パッカード入社．1990年エイ・ティー・アール通信システム研究所に出向．1994年横河・ヒューレット・パッカードへ復帰，現在に至る．プログラム仕様記述，言語処理系などの研究に従事．