PASCAL プログラム教授システムにおける誤り同定法

藤居藤樹† 渡邊豊英†
田中淳志† 杉江昇†

本論文ではプログラム言語 PASCAL を教授する知的 CAI システムにおけるプログラムの誤り同定法について述べる．すなわち，プログラムの記述問題に対して学習者の解答の正解，不正解を判断し，かつ不正解時には学習者の誤りを指示する方法を議論する．従来，知的 CAI におけるプログラムの診断機能は予想される様々な誤りに対する知識を用いて，誤りの原因を目標探索し，問題解決の手法を適用してきたが，多くの経験的知識を必要とし，教師が適時教材を追加することは一般に困難であった．本稿では，出題問題に対する模範解答をその解答のメタ知識として用いて効率的に対処できる方法を提案する．この方法では，新しい出題問題の追加も容易であり，一つの模範解答のみを用意すればよい．学習者の解答は，システムにあらかじめ保有されている模範解答と静的にプログラム構造を比較・照合されることにより，誤りが同定される．プログラムには様々な記述形式が可能であり，二つのプログラムが一見異なっていても誤りであるとは限らない．すなわち，制御構造，データ構造，演算式，文の並びなどに自由度があるためである．我々の方法では，変換規則を用いてプログラムを標準形式に変換し，さらに構文解析により中間コード系列に翻訳した後，ブロック単位の照合操作を段階的に実行することにより，誤り位置を同定する．

An Error Identification Method in the Tutoring System of PASCAL Programs

Touju Fujii,† Toyohide Watanabe,† Atsushi Tanaka† and Noboru Sugie†

An error identification function is one of the very important facilities to teach programming languages to students in the intelligent tutoring system. Traditionally, these types of functions are looked upon as a kind of problem solvers, which utilize much heuristic knowledge about predictive errors, in addition to the exact solution. However, such approaches are not always smart because the application ranges are very limited and the other new instructive exercises can not be easily added without the heuristic knowledge about the errors. We propose an experimental method to detect program errors and then identify the causes of the errors. Our approach is to compare student's answers with the system's solution from a static point of view. Our identification process is first to transform programs into the normal forms in source-to-source level, second to translate the source programs into the intermediate codes and then to analyze the corresponding blocks of programs by using a directed graph, stepwisely.

1. はじめに

知的教授システム (ITS: Intelligent Tutoring System) は，学習者個々人の理解状況，学習状態に合った個別指導の実現を目的に研究・開発されてきた[1)~3)]．学習者モデルによって学習者の理解状況を把握し，個別指導を実現する枠組みの下に様々な教授対象に対して種々のアプローチが報告されてきた[4)~6)]．これらの研究の多くは物理的な現象系，化学的な反応系，数学的な論理系を教授対象として専門知識教材を構成し，学習者の解答・質問に対して定性的な解釈・推論機構の下に対処する試みであった[7)~10)]．しかし，原理的な事実に基づいて種々の問題を論証できる場合を除いて，学習者の理解状況，学習状態に的確に対処することは必ずしも容易ではない．例えば，プログラミング教育では必ずしも唯一的なプログラムがあるわけではなく，様々な記述法・構成法があり，教授プロセスは複雑である[11)]．

本稿では，プログラム言語 PASCAL の初歩的なプログラミング教育を目的に，プログラムの誤り検出・同定法について報告する．PASCAL はアルゴリズム教育，プログラミング教育のプログラム言語として，計算機初歩教育の環境で用いられているが[12)]，プログ

† 名古屋大学工学部情報工学科
Department of Information Engineering, School of Engineering, Nagoya University

ラミングの初心者がその文法知識だけを前提に，正しいプログラム，効率的なプログラムを作成することは容易ではない．したがって，初心者のプログラムが正しいか否か，またプログラムに誤りがあればどこにそれが存在し，何がその原因であるかを検出・同定することは重要である．プログラムはアルゴリズム，データ構造，制御構造，演算処理，プログラム構造などの相違により様々な記述表現が可能である．見掛け上の記述表現が異なるために，二つのプログラムが同じ実行結果を生成しないとは判定できない．プログラムが正しいか否かを判定するには，静的な記述表現だけでなく，動的な振舞いによる検証が必要になる．しかし，動的な振舞いに対する解析は収集すべきデータが膨大になり，誤りの検出・同定はより困難になる．

我々のアプローチは静的にプログラムを解析し，それが正しいか否か，誤りがあればどこに原因があり，何が正しくないかを調べる．このとき，記述表現の多様性の問題を解決するために，

（1） プログラムを標準形式へ変換する

（2） あらかじめシステムに用意されている模範解答プログラムと学習者解答プログラムとの比較・照合はソース・プログラムではなく，中間コードで行う

（3） 演算順序の等価性を調べるために有向グラフを用いて処理する

（4） 変数，関数などの名前をあらかじめ出題中に指示した問題を扱う

ことによって対処した．（4）は本システムの制約ではあるが，必ずしも初歩的なプログラミング教育では制約とはいえない．プログラミング教育の入門書，解説書の演習問題・練習問題ではヒントの一部としてあらかじめプログラムの導入部を提示している場合が多い．

静的な解析法によって FORTRAN プログラムの誤りを検出する方法が LAURA で既に報告されている[13]．LAURA はソース・プログラムを有向グラフに変換し，有向グラフで演算式の簡約，文の並び順の判定を行っているが，この方法は非効率で，膨大な処理時間を要する．また，LAURA は非ブロック構造の FORTRAN に適用可能であるが，ブロック構造指向の PASCAL には容易に適用できない．

2. 処理の概要

学習者解答プログラムが正しいか否かを判定し，誤りがあれば何が原因で，どこに存在するかを識別するための方法として，あらかじめシステムに用意された模範解答プログラムと学習者解答プログラムを比較・照合する．もちろん，出題に応ずるプログラムは一通りではなく，様々な記述表現があり，プログラムを静的な記述表現だけから判断することはできない．我々のアプローチは静的にプログラム構造を解析し，二つのプログラムが処理手続きとして等価か否か，等価でなければどこが相違し，どのようにすれば等価になるかを検出・同定する．プログラムの記述表現の多様性の回避，誤り検出・同定の効率化のために，次のような特徴を有している．

1) 模範解答プログラムと学習者解答プログラムをそれぞれ標準形式に変換する

2) ソース・プログラムを中間コード系列に翻訳し，二つのプログラムを比較・照合する

3) 二つのプログラムの比較・照合は局所的に行うのではなく，プログラム全体を大局的に，かつ段階的にプログラムの分割を繰り返して実現される

4) 二つのプログラムの文集合，および文の並び順を

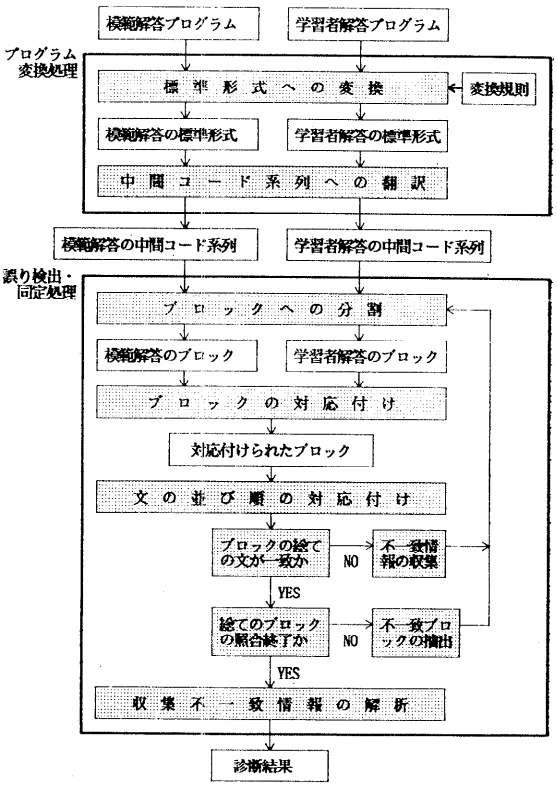


図 1 処理手続きの概略
Fig. 1 Processing flow.

```
2つの正整数が引数として与えられているとき、それら
の最大公約数を求める関数のプログラムを作れ。

program Euclid;
  var m,n:integer              ;2つの正整数
  function gcd(m,n):integer    ;最大公約数を求める関数
    var w:integer              ;テンポラリ変数
  begin
    2数の入力及び最大公約数の出力;
  end.
```

図 2 出題例
Fig. 2 An example of exercises.

調べるために，有向グラフを用いる

処理手順の概略を**図1**に示した．図1では，これらの特徴に対応する手続きが明示されている．1) と 2) は誤り検出・同定のための前処理として，記述表現の多様性の回避，比較・照合操作の効率化に対処する．もちろん，記述表現の多様性を発生する原因の一つに，変数・関数などの名前の問題があるが，我々の場合には**図2**に示す出題形式に問題を限定し，名前の問題を回避した．初歩的なプログラミング教育では，プログラムの記述法，構成法を学習させることが基本的な目的であり，大きな制約ではないと推定できること，また実際の教科書，手引書の練習問題・演習問題にも図2のような出題は数多いことに因る．

一方，3) と 4) は実際に二つのプログラムを比較・照合する方法に関係している．段階的にプログラムを有意なブロックに分割し，大域的なブロック間の相互関係と，局所的なブロック内の相互関係を定めて解釈することは，照合域を適切に限定でき，また処理の効率化を図ることができる．特に，ブロック構造の PASCAL では，このアプローチはプログラムそれ自身の構造に適している．

3. プログラムの変換

プログラムの変換処理では，様々な記述表現のプログラムを変換規則に従って書き換え，プログラム表現の統一化を図ることが目的で，(1)標準形式への変換，(2)中間コード系列への翻訳より成る．標準形式への変換はソース・プログラムからソース・プログラムへの書換え，また中間コード系列への翻訳はソース・プログラムから中間コードへの書換えである．これらの処理はコンパイラの作成に利用される技術とほぼ同じである[14]．

3.1 標準形式への変換

PASCAL をプログラミング教育の教授対象としているが，本研究ではそのフルセットではなく，扱うデータ型などに制約を課している．初歩的なプログラミング教育を目的とし，様々な応用プログラムを開発する専門的な技術教育ではないためであるが，プログラミング書法の本質的な学習という点で，フルセットでも，またサブセットでも相違はない．すなわち，

1) データ型：integer 型，boolean 型，array 型
2) 演算子：算術演算子 (+, −, *, /, div, mod),
論理演算子 (not, and, or),
関係演算子 (=, 〈〉, 〈=, 〉=, 〈, 〉)
3) 定義済手続き：read, write

に限定した．様々なプログラムの記述に豊富なデータ型の利用は必要不可欠であるが，基本的な問題に対して特に制約とはなっていない．もちろん，制御文など，種々の文が利用可能である．

標準形式への変換は，変換規則として用意した知識ベースに従って実行される．プログラムの基本的な書換えは次のようである．

1) 複合文：文と文を容易に対応付けるために，単純文の集まりとして表す．
2) 制御文：条件分岐文 (if 文，case 文)，繰返し文 (while 文，repeat 文，for 文)，無条件分岐文 (goto 文) は，else 文を伴わない if 文と goto 文のみのプログラムに変換する．これはまた，中間コードへの容易な翻訳処理を支援できる．例えば，

【例：if〜then〜else〜文の場合】

```
if 〈条件〉 then 〈文1〉 else 〈文2〉;
        ⇓(変換)
 if not (〈条件〉) then goto 〈ラベル1〉;
        〈文1〉;
〈ラベル1〉: if 〈条件〉 then goto 〈ラベル2〉;
        〈文2〉;
〈ラベル2〉: ……
```

である．この場合，ラベル1，ラベル2は自動的に生成される．付録1(a)には他の制御文に対する変換規則を整理した．

3) 演算式：算術式，論理式，関係式に対して，それぞれ変換規則を用いて書き換えるが，その変換手順の概略は以下のようである．この場合，算術式，論理式，関係式相互の変換順序はなく，演算式を走査して変換規則に該当すれば順次書き換える．

step 1：特定の演算子を書き換える．

step 2：不必要な括弧を除去する．
step 3：算術式を積和標準形式に，論理式を選言標準形式に変換する．また，関係式の場合，右辺を左辺に移項する．
step 4：演算式を簡約化する．
step 5：項を昇順に並び替える．

step 4 と step 5 はコンパイラのパーサ処理では必要とされないが，我々のパタン照合処理には照合処理の簡単化，正確さのために必要不可欠である．

【例：算術式における演算子 div (step 1)】
A div B⇒(A−(A mod B))/B

【例：算術式の非可換な二項演算子−，/(step 1)】
A−B⇒A+(−B)
A/B ⇒A∗(/B)
すなわち，反数 (−B)，逆数 (/B) とする．

【例：論理式における not の移動 (step 3, step 4)】
not (A or B) ⇒(not A) and (not B)
not (A and B)⇒(not A) or (not B)
not (not A) ⇒A
not True ⇒False
not False ⇒True

【例：関係式における右辺の左辺への移項 (step 3)】
A>B⇒A−B>0

付録1(b)には算術式における変換例を整理した．

3.2 中間コード系列への翻訳

標準形式に変換されたプログラムは，コンパイラの構文解析手法と同様に，中間コード系列へと翻訳される．中間コードは4組形式に準じて表現される．付録2(a)には，本研究で用いた中間コードを整理したが，この中間コードは付録2(b)の仮想スタック・マシンを想定して設定されている．

中間コードへ翻訳する目的は，二つのプログラムの比較・照合に際して照合対象を明示し，効率化を図るためである☆．すなわち，操作子をキー項として照合でき，文字列の照合に比べて処理を単純化できる．また，変数名，関数名，ラベル名などがトークン化されているために，処理が簡単である．さらに，中間コードにおける比較・照合は，ソース・プログラムによる比較・照合に比べて，変換規則による簡約化では吸収できなかった演算処理の多様な記述を，単純な中間コードの並びとすることにより，解消することも可能である．

4. プログラムの誤り検出・同定

単一操作子の並びである中間コード系列に翻訳されたプログラムは，それぞれ構造的な断片☆☆ を成すブロックに分割され，模範解答プログラムと学習者解答プログラムの分割ブロックがどのように対応し，また対応するブロックの中間コード系列が一致するか否かを照合して，プログラムに誤りがあるか否か，誤りがあればどこに存在し，何が原因かを調べる．プログラムの誤り検出・同定処理では，プログラム全体を直接比較・照合するのではなく，模範解答プログラムと学習者解答プログラムを構成するブロックを対応付けて，そのブロックごとに比較・照合する．

したがって，プログラムの比較・照合は局所的なブロック単位で実行され，処理の高速化・単純化を達成できる．ブロックに分割し，ブロックを比較・照合する処理は，比較・照合したブロックが不一致の場合に段階的に繰り返され，不一致のブロックが文に対応するレベルまで分割される．ブロック分割は基本的に中間コード系列で実行されるが，システム内には誤り情報指示のためにソース・プログラムも保存されており，ソース・プログラムと対応を図って，分割処理を容易にしている．

この誤り検出・同定処理は，(1)ブロックの分割，(2)ブロックの対応付け，(3)文の並び順の対応付けという手順で構成される．

4.1 ブロックの分割

ブロックは二つのプログラムを比較・照合する場合の処理単位であるが，段階的にその大きさは変化する．すなわち，ブロックは模範解答プログラムと学習者解答プログラムの照合操作で一致しなければ，さらに分割され，プログラム構造に従って順次，より小さなプログラムの断片になる．ただし，最小のブロックは文である．**図3**はプログラムのブロック分割の様子を表している．ハッチ部分が不一致のブロックを表し，それが段階的に細分化されている．**図4**は簡単なプログラムにおける文集合，および文をブロックとする分割例である．この例では，一回の分割で最小ブロックに至っている．図4ではソース・プログラムで

☆ LAURA[13]では，ソース・プログラムに対して演算式の簡約化 (reduction) を実行するために，その解釈・推定処理が複雑で，非効率となる．

☆☆ 関数，手続きなどの処理のまとまったものだけでなく，goto 文の後，ラベル付文の前などを分割の単位として，コンパイラの最適化におけるブロック分割と同様な処理で作成される[14]．

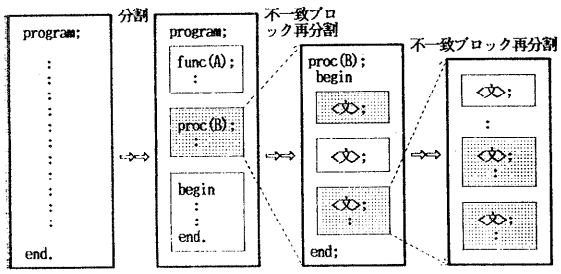


図 3 ブロックの段階的分割
Fig. 3 Multi-stages for block partition.

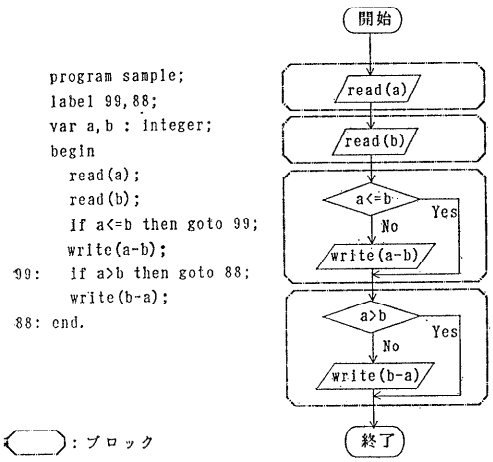


図 4 文集合，および文への分割例
Fig. 4 An example of block partition at statement-level.

分割例を示したが，実際には中間コード系列に対して適用される．分割方法を分かりやすくするために，ソース・プログラムで示した．したがって，4章の例はすべてソース・プログラムでその説明を示した．

4.2 ブロックの対応付け

模範解答プログラムと学習者解答プログラムはそれぞれブロックに分割されるが，ブロック内の中間コード系列の照合操作のために，これらのブロックを相互に対応付けられなければならない．プログラムによっては，あるブロックが他のブロックとは独立に実行できる場合があり，プログラム上のブロックの並びが正しい実行結果を生成するための唯一の並びではない．

一つのブロックがプログラム全体である場合には，ブロックの対応は一対一である．また，ブロックが副プログラムのときも，対応関係は名前によって一意に決定できる．ブロックが文集合，および文の場合，以下の基準によってブロックを対応付ける．

(a) ブロック内のすべての中間コード系列が一致する場合には，これらのブロックを対応付ける．

(b) ブロック内の中間コード系列が不一致である場合には，ブロック内の中間コードの数，文の数☆，手続き呼出し文の数，関数呼出しの数，if 文の数，goto 文の数などが等しいか，最も近いブロック同士を対応付ける．

(a)のように，すべてのブロックが明確に対応付けできれば，学習者解答プログラムは正しいプログラムと判定され，(b)のような場合には強制的に対応付けられたブロックをさらに文集合，および文まで段階的に分割し，再分割ブロック間で対応付けを繰り返す．

このように，ブロックを対応付けた結果，一方のプログラムのブロックが，対応付ける他方のプログラムに存在しない場合，誤りとしてそのブロックを抽出する．例えば，対応付けられないブロックが学習者解答プログラムに残っているとき，学習者解答プログラムに不必要な文集合，および文が存在するとして，また対応付けられないブロックが模範解答プログラムに残っているとき，学習者解答プログラムに必要な文集合，および文が欠除しているとして，誤りを発見できる．すなわち，ブロックの対応付けでは，必要な文の欠除や不必要な文の存在に関して誤りを検出・同定できる．

4.3 文の並び順の対応付け

一般に，二つのプログラムはそれを構成する文の順序が一致していなくても，同じ実行結果を出力することができる．プログラムが標準形式に変換されたとしても，文の実行順序に不確定性があり，実行の順序関係による文の並びを調べ，二つのプログラムが等価な実行結果を生成するか否かを判定する必要がある．

本研究では，模範解答プログラムの文の並びに対して，その実行結果を変えずに，文の実行順序を変更して，学習者解答プログラムの文の並びに対応付けることができるかを調べる．模範解答プログラムを変形して，学習者解答プログラムに接近する方法は，学習者解答プログラムを変形して模範解答プログラムに接近する方法よりも，変形操作が制約されるために，容易である．

(1) 文の実行順序の前後関係

二つのプログラムに対して，これらの文の並びを同じ順序に変更できるか否かを調べる．これはコンパイ

☆ 中間コードでは格納命令を文の終了として，格納命令の数で文の数を換算する．

ラの最適化技術とほぼ同じアプローチとなる[14]．文の実行関係は，

（a） 制御の流れ

（b） 変数値の変化

について検討する必要がある．しかし，制御の流れに対して，実行時の順序関係を保持するようにブロック（文集合，および文）を分割したために，ブロック間では実行順序が保証されている．したがって，変数の依存関係の下に文の並び順を検討する．

変数の定義と使用を以下のように定める．

【変数の定義】

1. 代入文の左辺の変数
2. 入力文 read の引数

【変数の使用】

1. 代入文の右辺の変数
2. 演算式の変数
3. 手続き呼出し文の引数
4. 関数呼出しの引数
5. 出力文 write の引数

変数の定義は変数に値を設定する地点を明示し，変数の使用は変数の値を利用する地点を明示している．したがって，ある変数について考えると，変数を定義する文のブロックが前ブロック，変数を使用する文のブロックが後ブロックとして前後関係が定められる．

このように，すべての変数についてその定義位置とその使用位置を調べ，すべてのブロックの前後関係を定めることができる．

（2） 文の並替え条件

プログラムの実行結果を変えずに，文を並び替えるには，並び替える文において変数の定義と使用の関係が変化しなければよい．すなわち，任意の二つの文に着目して，

（a） 二つの文の前後関係が変化しない

（b） 二つの文の間にその変数を再定義する文が挿入されない

という条件の下で並び替えればよい．この並替えの条件を調べるために，有向グラフを用いる．

有向グラフのノードはブロック（文集合，および文）を，またエッジはブロック間の前後関係を表す．この有向グラフはブロックの分割と対応付けによって生成されたブロック間の前後関係を形式化した表現となっている．

ブロックの並び順の変更手続きは，ノード間の接続関係を調べ，矛盾がなければエッジを変更する．ノード間の接続関係を調べるために，エッジには以下のブロック情報を割り付ける．ブロック情報とは，

- エッジを通るすべての前ブロック
- エッジを通るすべての前ブロックに含まれる変数を再定義する後ブロック

を除いたブロック名の集合である．

以上のことから，プログラムの実行結果を変えない，すべての文の並びを有向グラフの経路として列挙できる．列挙された経路に，学習者解答プログラムのブロックの並び順と等しい経路が存在すれば，学習者解答プログラムは正しい．そうでなければ，学習者解答プログラムに誤りがあるとして，学習者解答プログラムのブロックの並び順に最も近い経路を一つ選択し，それに基づいて誤りを検出・同定する．例えば，ブロック1からブロック5の間に，

〔例〕 ブロック2＜ブロック4
ブロック2＜ブロック5
ブロック3＜ブロック4
ブロック3＜ブロック5

という前後関係が存在すると仮定する．ただし，記号＜は「X＜Y」においてXがYの前で，YがXの後という関係を表す．このとき，有向グラフは**図5**のようにノードとエッジが構成され，それぞれのエッジに

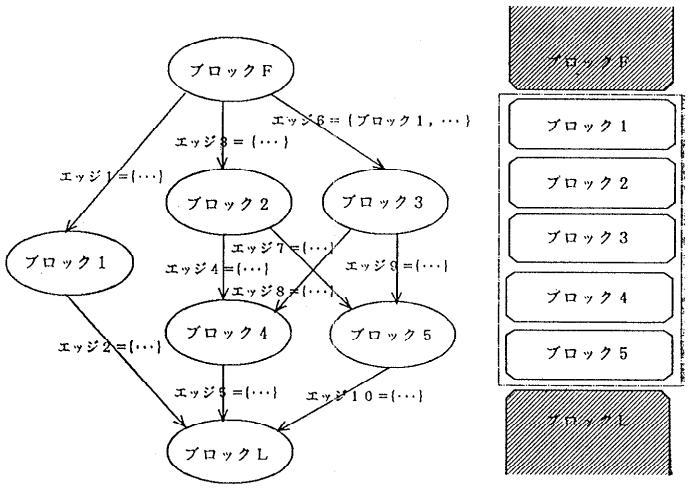


図5 ブロックの並び順を表す有向グラフ
Fig. 5 Directed graph for relationships among blocks.

ブロック情報が割り当てられる．

図5で，ブロックF，Lは次のように定義される．

- ブロックF：現在並び替えるブロックの集合よりも前に位置するブロック
- ブロックL：現在並び替えるブロックの集合よりも後に位置するブロック

(3) ブロックの並替え手続き

ブロックの並替え手続きを具体例に従って説明する．**図6**に，一連の処理手順を示した．ただし，図6では処理内容を分かりやすくするために，ソース・プログラムを用いているが，実際には中間コード系列のコード照合操作で一致か不一致かを調べる．

図6(a)に示したプログラムは，入力された2数の差の絶対値を出力する．学習者解答プログラムは模範解答プログラムと並び順は異なるが，模範解答プログラムと同じ実行結果を出力し，正解である．

ブロックの分割，および対応付けの結果を図6(b)に示す．ここで，学習者解答プログラムのブロックと等しく並べようとすると，模範解答プログラムのブロックは $BT_2 \rightarrow BT_1 \rightarrow BT_4 \rightarrow BT_3$ の順に並んでいる必要がある．模範解答プログラムのブロックの前後関係を調べると，BT_1 で定義された変数 a が BT_3 と BT_4 で使用され，BT_2 で定義された変数 b が BT_3 と BT_4 で使用されているので，図6(c)に示す有向グラフを構成できる．さらに，各エッジにブロック情報を割り付けると，図6(d)に示す有向グラフとなる．次に，学習者解答プログラムと等しい並び順の経路を一つ選択するが，ここでは $BT_1 \rightarrow BT_3$,

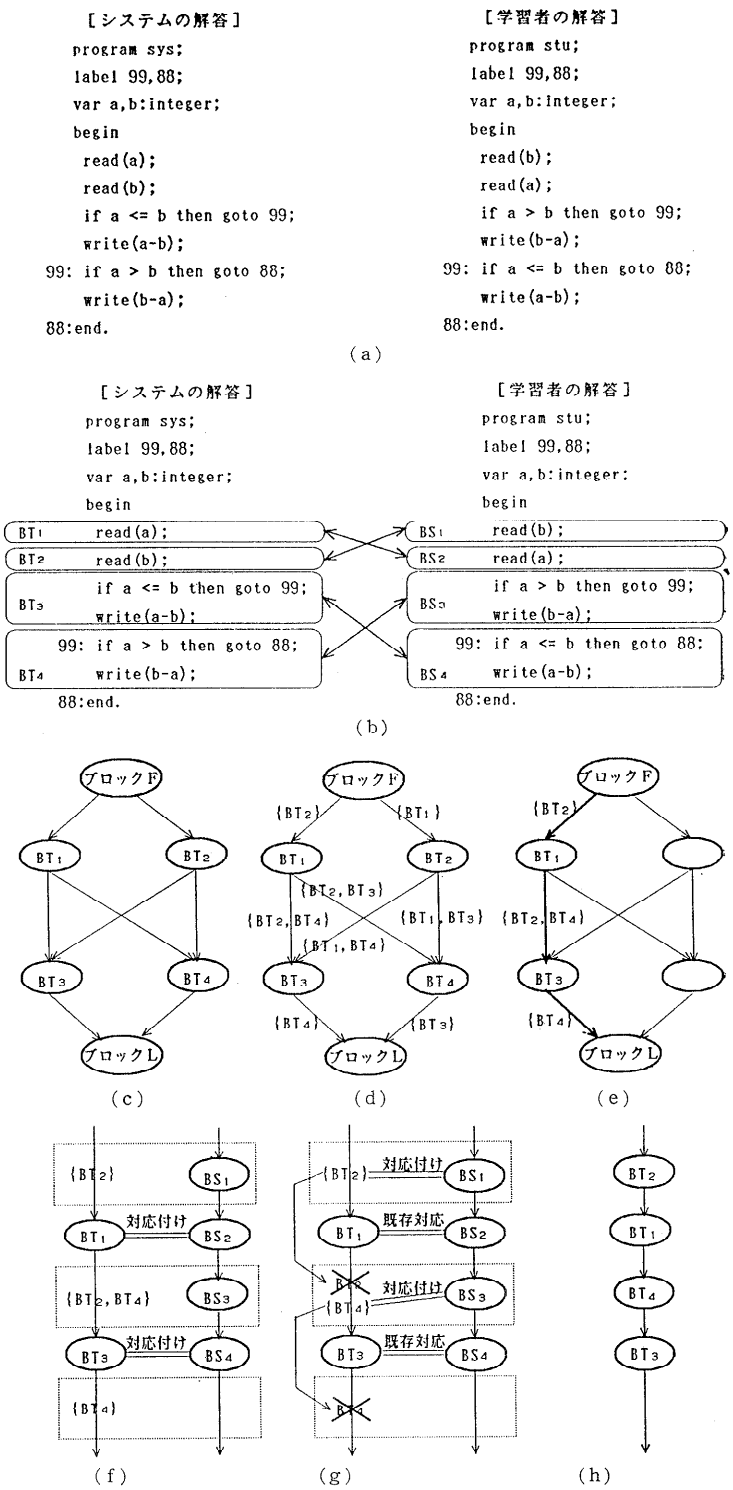


図 6 並換えの具体例

Fig. 6 An example of block correspondence process.

$BT_1 \rightarrow BT_4$, $BT_2 \rightarrow BT_3$, $BT_2 \rightarrow BT_4$ のどの経路も，対応する学習者解答プログラムの並び順と等しいので，番号の最も小さい $BT_1 \rightarrow BT_3$ を選択する．この結果が図6(e)である．

BT_1, BT_3 のブロックの位置が決定されたので，BT_2 と BT_4 の位置を決定する．図6(f)により，BT_2 は BT_1 より前か，BT_1 と BT_3 の間に置かれる可能性があり，BT_4 は BT_1 と BT_3 の間か，BT_4 より後に置かれる可能性がある．したがって，模範解答プログラムの有向グラフの経路と学習者解答プログラムのブロックの並びを比較すると，BT_1 と BS_2, BT_3 と BS_4 が対応するので，BT_1 より前に BS_1 に対応する模範解答プログラムのブロックが置かれ，BT_1 と BT_3 の間に BS_3 に対応する模範解答プログラムのブロックが置かれれば，学習者解答プログラムと同じ順が得られ，並替え可能となる．図6(g)から，BT_2 と BS_1 は対応し，BT_4 と BS_3 は対応するので，$BT_2 \rightarrow BT_1 \rightarrow BT_4 \rightarrow BT_3$ という学習者解答プログラムのブロックの並びと等しい模範解答プログラムのブロックの並びが得られる．この結果，模範解答プログラムのブロックの並びは図6(h)となり，学習者解答プログラムは正解と判定される．

5. 診 断 例

前章までに述べた方法に基づいて，PASCAL プログラミング教育用の教授システムをワークステーション UNISYS/KS 301 で，LISP を用いて実現した．例題として**図7**に示した問題を出題し，**図8**のような解答プログラムを学習者が作成したとする．ただし，模範解答プログラムは**図9**である．図8と図9を比べると，制御文，演算式，文の並び順などに違いがあるが，実際の誤りは「fac:=1;」が欠除しているだけである．

模範解答プログラムと学習者解答プログラムを標準形式に変換し，中間コード系列に翻訳した結果が**図10**である．そして，この中間コード系列をブロックに分割し，分割ブロックを対応付け，文の並び順を対応付けて誤りを検出・同定した結果が**図11**である．この診断例では，模範解答プログラムのすべての文を学習者解答プログラムに対応付けることができず，

```
問題
問題5-2-1
 1から10までの数の階乗の値を求めよ

program factorial;
var n:integer  ;1から10までの数
    i:integer  ;テンポラリ変数
    fac:integer;階乗の結果
begin
 1から10までの数およびその階乗の値を出力

end
```

図7 出題問題
Fig. 7 Another exercise.

```
program factorial;
var n,i,fac:integer;
begin
  n:=1;
  while n<=10 do
    begin
      for i:=1 to n do
        fac:= i*fac;
      write(fac);
      write(n);
      n:=n+1
    end
end.
```

図8 学習者解答プログラム
Fig. 8 Program of student.

```
program factorial;
var n,i,fac:integer;
begin
  for n:=1 to 10 do
    begin
      fac:=1;
      for i:=1 to n do
        fac:= fac*i;
      write(n);
      write(fac)
    end
end.
```

図9 模範解答プログラム
Fig 9 Program of system.

```
CODE-OF-SYSTEM

(0 INT * * 4)
(1 LIT * * 1)
(2 STO 0 0 1)
(3 LIT * * 10)
(4 LOD 0 0 1)
(5 REV. * * *)
(6 ADD * * *)
(7 LIT * * 0)
(8 LSS * * *)
(9 JPC * * 2)
(10 JMP * * 31)
(11 LIT * * 1)
(12 STO 0 0 3)
(13 LIT * * 1)
(14 STO 0 0 2)
(15 LOD 0 0 1)
(16 LOD 0 0 2)
(17 REV. * * *)
(18 ADD * * *)
(19 LIT * * 0)
(20 LSS * * *)
(21 JPC * * 2)
(22 JMP * * 10)
(23 LOD 0 0 3)
(24 LOD 0 0 2)
(25 MUL * * *)
(26 STO 0 0 3)
(27 LIT * * 1)
(28 LOD 0 0 2)
(29 ADD * * *)
(30 STO 0 0 2)
(31 JMP * * -16)
(32 LOD 0 0 1)
(33 PUT * * *)
(34 LOD 0 0 3)
(35 PUT * * *)
(36 LIT * * 1)
(37 LOD 0 0 1)
(38 ADD * * *)
(39 STO 0 0 1)
(40 JMP * * -37)
```

```
CODE-OF-STUDENT

(0 INT * * 4)
(1 LIT * * 1)
(2 STO 0 0 1)
(3 LIT * * 10)
(4 LOD 0 0 1)
(5 REV. * * *)
(6 ADD * * *)
(7 LIT * * 0)
(8 LSS * * *)
(9 JPC * * 2)
(10 JMP * * 29)
(11 LIT * * 1)
(12 STO 0 0 2)
(13 LOD 0 0 1)
(14 LOD 0 0 2)
(15 REV. * * *)
(16 ADD * * *)
(17 LIT * * 0)
(18 LSS * * *)
(19 JPC * * 2)
(20 JMP * * 10)
(21 LOD 0 0 3)
(22 LOD 0 0 2)
(23 MUL * * *)
(24 STO 0 0 3)
(25 LIT * * 1)
(26 LOD 0 0 2)
(27 ADD * * *)
(28 STO 0 0 2)
(29 JMP * * -16)
(30 LOD 0 0 3)
(31 PUT * * *)
(32 LOD 0 0 1)
(33 PUT * * *)
(34 LIT * * 1)
(35 LOD 0 0 1)
(36 ADD * * *)
(37 STO 0 0 1)
(38 JMP * * -35)
```

図10 標準形式変換後の中間コードへの翻訳結果
Fig. 10 Intermediate codes.

```
診断結果
**********************************************

答案を受け取りました

 program factorial;
 var n,i,fac:integer;
 begin
   n:=1;
   while n<=10 do
     begin
       for i:=1 to n do
         fac:= i*fac;
       write(fac);
       write(n);
       n:=n+1
     end
 end.

あなたの答案を診断します

**********************************************

fac: =1 ;
が,以下の文の前に必要です
for i: =1  to n do
```

図 11 誤り検出・同定結果
Fig. 11 Execution result.

「fac :=1 ;」が必要であることが通知されている.

6. おわりに

本稿では，PASCAL プログラムの教授システムについて誤り検出・同定処理を検討した．その処理は，記述表現の多様性を回避するために，標準形式に変換し，中間コード系列に翻訳する前処理を実施した後，プログラムの断片であるブロックへの分割，そのブロックの対応付け，文の並び順の対応付けという手順で実現される．このとき，文の並び順の対応付けのために有向グラフを用いた．

本稿のような目的の下に誤り検出・同定処理を報告している研究に LAURA がある[13]．LAURA は FORTRAN に対して検討しているが，本研究とは異なってソース・プログラムを直接の処理対象とし，ソース・プログラム全体を有向グラフに変換し，有向グラフ上での演算式の簡約，文の並び順の対応付けを試みている．これは，FORTRAN のような単一構造のプログラムには有効であると思われる．しかし，ブロック構造（入れ子構造）の PASCAL では，入れ子関係を効果的に把握する必要がある．LAURA ではプログラム構造が表現されず，対応・照合が全プログラム域でなされ，非効率となる．LAURA と比べて，我々の方法は次の点に特徴がある．

1) LAURA は変数名，関数名に制約を課していないが，名前の同定・推定処理を必要とし，これが完全に遂行されるという保証はない．例えば，FORTRAN では空白は無視される．したがって，構文解析を実行しないと，回避できない問題となる．一方，我々の方法は名前に制約を課しているが，実際には中間コードで対応・照合するために実質的に名前の解釈に困難さがない．

2) LAURA も有向グラフを用いているが，我々の場合，ブロック単位に有向グラフを作成し，その間でノードの対応付けを図り，探索域を限定するために処理量は少ない．一方，LAURA はプログラム全域で有向グラフを作成するために探索域が広く，非効率である．

3) LAURA の方法は非ブロック構造のプログラム言語に対してソース・プログラムからの有向グラフを文単位に構成できるが，ブロック構造のプログラム言語に対して単一の有向グラフを構成すると，プログラム構造を保存できず，無駄な解釈が必要である．すなわち，LAURA を直接適用すると，効率良く処理するために階層的な有向グラフを扱う機構が必要となる．一方，我々の方法はブロック分割が段階的に実施され，必要に応じてブロックごとに有向グラフを作成し，処理するので，汎用性があり，効率的である．

4) LAURA では演算式の簡約，文の並び順の並替えを有向グラフのノードに対応する文で処理し，パタン照合の処理量が多いが，我々の方法ではソース・プログラムを標準形式に変換し，かつ中間コード系列へ翻訳するので，パタン照合は必要なく，中間コードの操作子を基本とした照合となるので，効率良く実行できる．

したがって，より汎用的，効率的な方法である．また，出題問題も教師が適時作成し，システムに用意できるなどの特徴もある．

今後の課題として，

（1） 初歩的なプログラミング教育として，言語使用を制約したが，これを緩めて，より多様なプログラムを処理対象とする

（2） 出題形式の制約を緩めて，より多くの練習問題・演習問題を処理対象とする

など，適用域の拡大が必要である．

謝辞 日頃よりご指導いただいている名古屋大学工学部・稲垣康善教授，鳥脇純一郎教授，ならびに中京大学情報科学部長・福村晃夫教授に深謝します．ま

た，熱心に討論していただいた研究室の皆様，およびいろいろと配慮いただいている杉野花津江氏に感謝します．さらに，有益な指摘・意見をいただいた査読者の方に感謝します．

参考文献

1) Sleeman, D. and Brown, J. S.: *Intelligent Tutoring Systems*, Academic Press, Inc. (1982).
2) Wenger, E.: *Artificial Intelligence and Tutoring Systems*, p. 432, Morgan Kaufman Pub. Inc. (1987).
3) 大槻説乎，山本米雄：知的 CAI のパラダイムと実現環境，情報処理，Vol. 29, No. 11, pp. 1255-1274 (1988).
4) 池田 満，溝口理一郎，角所 収：学生モデル記述言語 SMDL と学生モデルの帰納推論アルゴリズム，電子情報通信学会論文誌 D-II，Vol. J 72-D-II，No. 1, pp. 112-120 (1989).
5) 中村祐一，平島 宗，上原邦昭，豊田順一：ITS のための説明機能に関する検討—ドメイン原理および類推を用いた学生モデル生成法—，電子情報通信学会論文誌 D-II，Vol. J 73-D-II，No. 3, pp. 399-407 (1990).
6) 竹内 章：知的 CAI のフレームワーク BOOK の構成といくつかの応用例，情報処理，Vol. 29, No. 11, pp. 1309-1315 (1988).
7) 小西達裕，伊藤幸宏，高木 朗，小原啓義：ICAI における知識の成立原理の教示と対象世界のシミュレーション，電子情報通信学会論文誌 D-II，Vol. J 73-D-II，No. 7, pp. 1007-1028 (1990).
8) 渡辺成良，渋沢良裕：スクリプトに基づいた個別指導：回路解析のスキル獲得のための知的 CAI における直接的指導方略，人工知能学会誌，Vol. 6, No. 6, pp. 912-919 (1991).
9) 中村祐一，平島 宗，上原邦昭，豊田順一：学生の誤りを解釈する機構の開発—ITS のための一般化および類推を用いたモデル生成法—，電子情報通信学会論文誌 D-II，Vol. J 72-D-II，No. 7, pp. 1077-1087 (1989).
10) 平島 宗，中村祐一，上原邦昭，豊田順一：認知的考察に基づく知的 CAI のための学生モデルの生成法—プロセス駆動型モデル推論法—，電子情報通信学会論文誌 D-II，Vol. J 73-D-II，No. 3, pp. 408-417 (1990).
11) 加藤 等，田中淳志，渡辺豊英，吉田雄二：LISP—CAI システムにおける診断と計画機能，情報処理学会コンピュータと教育研究会資料，11-2 (1990).
12) 米田信夫，疋田輝雄：PASCAL プログラミング，p. 151，サイエンス社 (1979).
13) Adam, A. and Laurent, J.-P.: LAURA, A System to Debug Student Programs, *Artif. Intell.*, Vol. 15, pp. 75-122 (1980).
14) 中田育夫：コンパイラ，p. 272，産業図書 (1981).

付 録

1（a） 制御文の標準形式

（1） case 文

```
case 〈式〉 of 〈タグ1〉: 〈文1〉;……; 〈タグN〉:
〈文N〉 end;
  ⇩変換
  if 〈式〉 ◇ 〈タグ1〉 then goto 〈ラベル1〉;
    〈文1〉;
〈ラベル1〉: ……
          :
  if 〈式〉 ◇ 〈タグN〉 then goto 〈ラベルN〉;
    〈文N〉;
〈ラベルN〉: ……
```

（2） while 文

```
while 〈条件〉 do
    begin 〈文1〉; ……; 〈文N〉 end;
  ⇓変換
〈ラベル1〉: if not (〈条件〉) then goto 〈ラベル
                2〉;
              〈文1〉; ……; 〈文N〉; goto 〈ラ
              ベル1〉;
〈ラベル2〉: ……
```

（3） for 文

```
for 〈変数〉:=〈式1〉 to 〈式2〉 do
    begin 〈文1〉; ……; 〈文N〉 end;
  ⇓変換
〈ラベル1〉: if 〈変数〉>〈式2〉 then goto 〈ラ
              ベル2〉;
              〈文1〉; ……; 〈文N〉; 〈変数〉:=
              〈変数〉+1; goto 〈ラベル1〉;
〈ラベル2〉: ……
```

〔注意〕 “to” が “downto” であれば変換後の文は以下の点で異なる．

1) if 文の不等号が “<” となる．
2) 制御変数のインクレメントがデクレメントになる．

（4） repeat 文

```
repeat 〈文1〉; ……; 〈文N〉;
  until 〈条件〉;
  ⇓変換
〈ラベル1〉: 〈文1〉; ……; 〈文N〉;
            if 〈条件〉 then goto 〈ラベル2〉;
            goto 〈ラベル1〉;
〈ラベル2〉: ……
```

1(b)-1 演算式の標準形式への変換例

(1) 不必要な括弧の省略

〔例〕 (A*B)*C ⇒(変換) A*B*C
(A+B)+C ⇒(変換) A+B+C

(2) 分配則による積和標準形式への変換

〔例〕 A*(B+C) ⇒(変換) A*B+A*C

(3) 単項演算子 −, / の移動と簡約

〔例〕 (−(A+B)) ⇒(変換) (−A)+(−B)
(/(A*B)) ⇒(変換) (/A)*(/B)
(−A)*B ⇒(変換) (−A*B)
(−(−A)) ⇒(変換) A
(/(/A)) ⇒(変換) A
(−(/A)) ⇒(変換) (/(−A))

(4) 定変数の定数置換え

〔例〕 const pai=3.14; const pai=3.14;
: ⇒(変換) :
x:=2*pai*r; x:=2*3.14*r;

(5) 定数計算が可能な部分の簡約化

〔例〕 A*(5+2*3) ⇒(変換) 11*A

(6) 算術式の簡約化

〔例〕 A*1 ⇒(変換) A
A*0 ⇒(変換) 0
A+0 ⇒(変換) A
(−A)+A ⇒(変換) 0
(/A)*A ⇒(変換) 1

(7) 交換則による項の昇順への書換え

〔例〕 B*A ⇒(変換) A*B
B+A ⇒(変換) A+B

1(b)-2 論理式の標準形式への変換例

(1) 不必要な括弧の省略

〔例〕 (A or B) or C ⇒(変換) A or B or C
(A and B) and C⇒(変換) A and B and C

(2) 分配則による選言標準形式への書換え

〔例〕 A and (B or C) ⇒(変換) (A and B) or (A and C)

(3) 簡約化

〔例〕 A or True ⇒(変換) True
A and True ⇒(変換) A
A or False ⇒(変換) A
A and False ⇒(変換) False
A or (not A) ⇒(変換) True
A and (not A) ⇒(変換) False

(4) 交換則による項の昇順並替え

〔例〕 B or A ⇒(変換) A or B
B and A ⇒(変換) A and B

1(b)-3 関係式の標準形式への変換例

(1) 同値式による not の除去

〔例〕 not (A=0) ⇒(変換) A◇0
not (A◇0) ⇒(変換) A=0
not (A<0) ⇒(変換) A>=0
not (A>0) ⇒(変換) A<=0
not (A<=0) ⇒(変換) A>0
not (A>=0) ⇒(変換) A<0

(2) 同値式による関係子 ◇, <=, >=, > の除去

〔例〕 A◇0 ⇒(変換) (A<0) or ((−A)<0)
A<=0 ⇒(変換) (A<0) or (A=0)
A>=0 ⇒(変換) ((−A)<0) or (A=0)
A>0 ⇒(変換) (−A)<0

(3) 式の簡約化

〔例〕 2*X+4*Y>0 ⇒(変換) X+2*Y>0

2(a) 中間コード

(1) Load literal: (lit, *, *, a)

〔意味〕 R2:=R2+1; s〔R2〕:='a'

(2) Load: (lod, b, i, a)

〔意味〕 R2:=R2+1; if i=R4 then s〔R2〕:=s〔R1(b)+R4+a〕;
if i=0 then s〔R2〕:=s〔R1(b)+a〕
ただし, bは Static Line をさかのぼる回数

(3) Store: (sto, b, i, a)

〔意味〕 R2:=R2+1; if b=R2 then
if i=R3 then s〔R2+R3+a〕:=s〔R2〕;
if i=0 then s〔R2+a〕:=s〔R2〕;
if b='number' then if i=R3 then
s〔R1(b)+R3+a〕:=s〔R2〕;
if i=0 then s〔R1(b)+a〕:=s〔R2〕;
ただし, bは Static Line をさかのぼる回数

(4) Store index register: (sti, *, i, *)

〔意味〕 Ri:=s〔R2〕; R2:=R2−1

(5) Arithmetic operation:

- (rev, *, *, *): s〔R2〕:=−s〔R2〕
- (inv, *, *, *): s〔R2〕:=1/s〔R2〕
- (add, *, *, *): R2:=R2−1; s〔R2〕:=s〔R2〕+s〔R2+1〕
- (mul, *, *, *): R2:=R2−1; s〔R2〕:=s〔R2〕*s〔R2+1〕
- (mod, *, *, *): R2:=R2−1; s〔R2〕:=s〔R2〕 mod s〔R2+1〕

・(eql, ＊, ＊, ＊): R2:=R2−1;
if s〔R2〕=s〔R2+1〕
then s〔R2〕:=1 else
s〔R2〕:=0

・(les, ＊, ＊, ＊): R2:=R2−1;
if s〔R2〕<s〔R2+1〕
then s〔R2〕:=1 else
s〔R2〕:=0

・(and, ＊, ＊, ＊): R2:=R2−1; if s〔R2〕=1
and s〔R2+1〕=1
then s〔R2〕:=1 else
s〔R2+1〕:=0

・(or, ＊, ＊, ＊): R2:=R2−1; if s〔R2〕=1
or s〔R2+1〕=1
then s〔R2〕:=1 else
s〔R2+1〕:=0

・(not, ＊, ＊, ＊): if s〔R2〕=1 then s〔R2〕
:=0;
if s〔R2〕=0 then s〔R2〕
:=1

(6) Increment stack top: (int, ＊, ＊, a)
〔意味〕 R2:=R2+a

(7) Procedure call: (pcal, ＊, ＊, a)
〔意味〕 s〔R2+1〕:=R1; s〔R2+2〕:=PC;
R1:=R2+1; PC:=a

(8) Function call: (fcal, ＊, ＊, a)
〔意味〕 s〔R2+1〕:=R1; s〔R2+2〕:=PC;
R1:=R2+1; RC:=a

(9) Return: (rtn, ＊, ＊, ＊)
〔意味〕 R2:=R1−1; PC:=s〔R1+1〕;
R1:=s〔R1〕

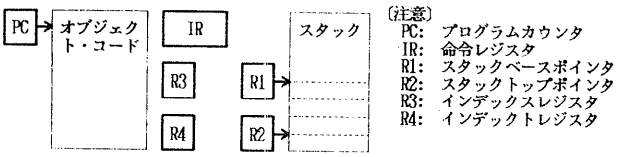
(10) Unconditional jump: (jmp, ＊, ＊, a)
〔意味〕 PC:=PC+a

(11) Jump on condition: (jpc, ＊, ＊, ＊)
〔意味〕 if s〔R2〕=0 then PC:=PC+a; R2:=
R2−1

(12) Read data: (get, ＊, ＊, ＊)

(13) Write data: (put, ＊, ＊, ＊)

2(b) 仮想スタック・マシン



(備考) 各レジスタの初期値: PC=0, R1=0, R2=−1, R3=0, R4=0

・s〔R〕はレジスタRが指示する番地の内容を表す.

(平成3年4月18日受付)
(平成4年12月10日採録)

藤居 藤樹
1968年生. 1991年名古屋大学工学部情報工学科卒業. 同年オムロン(株)システム総合研究所入社. 以来, 言語処理系の研究・開発に従事. 在学中には知的CAIの研究を行った.

渡邉 豊英 (正会員)
1948年生. 1972年京都大学理学部修了. 1974年同大学院工学研究科数理工学専攻修士課程修了. 1975年同博士課程中途退学. 同年京都大学大型計算機センター助手. 1987年名古屋大学工学部情報工学教室助教授. 京都大学工学博士. 統合化環境, 分散協調環境, データベース環境, データベースの高度インタフェース, 知的CAI, 文書理解, 地図理解に興味を持つ. 電子情報通信学会, 日本ソフトウェア科学会, 人工知能学会, ACM, IEEE Computer Society 各会員.

田中 淳志 (正会員)
1967年生. 1990年名古屋大学工学部情報工学科卒業. 1992年同大学院工学研究科情報工学専攻博士・前期課程修了. 現在松下電器産業(株)中央研究所電子機器基礎研究所勤務. マルチメディア, データベース・システムなどの研究に従事. 在学中, 知的CAIの研究を遂行.

杉江 昇 (正会員)
昭和7年生. 昭和32年名古屋大学工学部電気工学科卒業. 電子技術総合研究所バイオニクス研究室長, 視覚情報研究室長を経て, 現在名古屋大学工学部情報工学科教授. 視覚情報処理, 神経科学, 自然言語などの教育・研究に従事. 工学博士.