知識工学と人口知能　68－6
（１９９０．１．１８）

プログラム理解に基づくプログラム診断システム

海尻 賢二
信州大学工学部

概要

プログラミング教育においてはプログラミング演習は不可欠であり、これを知的にサポートするためにプログラムの理解及び理解に基づく診断という機能が必要となる。問題の段階的詳細化設計法に従った［問題－設計－ゴール－プラン－テンプレート］という階層に基づいた問題およびその解法の記述法を提案し、この記述法に基づいて初心者のｐａｓｃａｌプログラムを診断するシステム（PascalTutor）を実現中である。プログラム理解における大きな問題は与えられた問題に対する多くの variation の取り扱いである。PascalTutor ではこの variation を上記の階層に応じて分類し、各レベルの variation として階層的に認識する。

Program Diagnosis Based on Program Understanding

Kenji kaijiri
Faculty of Engineering,Shinshu University
500 Wakasato,Nagano 380,Japan

December 4, 1989

Abstract

Program understanding is an important part of the domain expertise required for intelligent tutoring systems that teach programming. We explored the process by which students implement programs, and developed a program diagnosing system based on program understanding. The understanding of student programs is greatly complicated by the tremendous variability that arises in student solutions even to trivial tasks. We cope with this variability by hierarchical problem description.

1 まえがき

ソフトウェア技術者の養成が急務となっている現在、プログラミング教育のための知的ＣＡＩシステムの必要性は強く認識されつつある。プログラミング教育においてプログラミング演習は不可欠であり、これを知的にサポートするためにはプログラムの理解及び理解に基づく診断という機能が必要となる。プログラム自体がいくつかの階層構造の上に成り立っているため、その理解も次のような階層に基づいて行なう必要がある。

1. 構文レベル
2. 静的意味レベル
3. 動的意味レベル

このうち（１）と（２）はいわゆるバグの知的な診断という範疇に属するものであり、基本的には言語のみに基づいて診断が可能である。（（３）と関連づけて全体として判断したほうが好ましいが独立に考えてもそこそこの診断は可能である）

（３）はプログラムが対象としている問題に依存するものであり、一般的には命題等を利用しないとその診断はできない。しかし本論で対象とするのは初心者のプログラムの理解であり、プログラムの検証等でとられている命題挿入によるアプローチは採れない。そこで何らかの形式での問題の記述（またはプログラムの記述）に基づいて対象プログラムを理解することが必要となる。このような形式でのプログラム診断についてはこれまでいくつかの研究が行なわれている。

［ＬＡＵＲＡ］[1]ではプログラムグラフという形式で正しいプログラムを表現し、それと学生プログラムとの照合をとることにより正否の判定及び診断を行なう。但し単なるテキスト上での照合だけでは種々の variation が吸収できないため、変換規則を数多く用意し、学生プログラムをできるだけ標準形に変換した後に照合を行なう。標準プログラムをベースとするため意味的に異なるもの（考え方の異なるもの）は扱えない。

［ＰＲＯＵＳＴ］[2]では上記の問題点を考慮して、個々の問題に対する正しいプログラム（標準的な誤ったプログラムも）をｇｏａｌ－ｐｌａｎというものの組合せで表現する。基本的には問題独立なｐｌａｎ集合を多数用意しておき、そのｐｌａｎにもとづいたｇｏａｌの組合せとして問題を記述する。ｇｏａｌに対してそれを実現するするｐｌａｎの variation 及び問題のｇｏａｌ分割の variation によりプログラムの variation を吸収しようとしている。またｐｌａｎの variation とも言えないような小さな相違点は［ＬＡＵＲＡ］と同様変換規則を用意して対処している。プログラムの記述をｐｌａｎという比較的

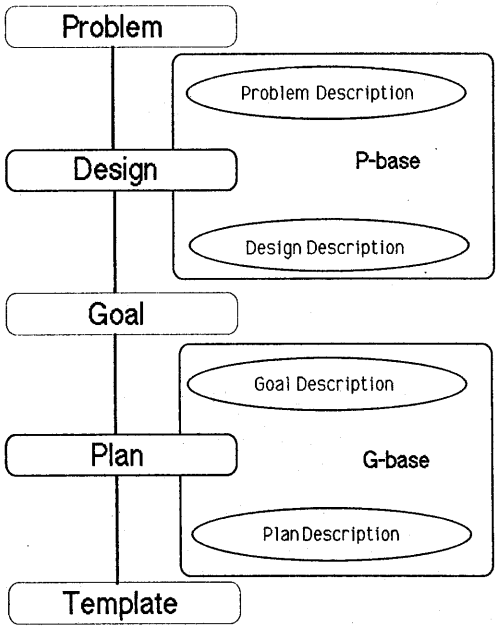

図 1: 問題記述の階層性

小規模な構造に限定しているために、ある程度の規模のプログラム（ｔｏｙプログラムにしても）を記述するのが煩雑になるという欠点を持つ。

［ＬｉｓｐＴｕｔｏｒ］[3]では学生の問題解決と同時進行的に診断を行なう。すでにできているプログラムの診断という立場はとっていない。

［INTELLITUTOR］[4]では手続きグラフという形式の問題表現を提案している。手続きグラフでは上部では抽象的な手続き的概念による分割、下部ではｐｌａｎ的なものによる分割を行なっている。

［Ｔａｌｕｓ］[5]では対象を LISP とし、種々の LISP 関数に対するアサーションの証明により、プログラムの等価性、またその相違点を判定する。プログラムの論理に基づいて等価性を判定するので等価性がその手続きの中に埋め込まれてしまい、拡張が困難であり、また設計の違いといった事の取り扱いも難しい。

そこでわれわれは基本的には goal-plan の考えに基づき、段階的詳細化に基づくプログラム設計を辿るという形式でのプログラム理解を目的として、問題に対する解を（問題－設計－ゴール－プラン－テンプレート）という階層で記述するという方式（図１参照）を新たに提案し、この方式に基づき、初心者の pascal プログラムを診断するシステム（PascalTutor）を実現中である。PascalTutor では上記の各階層の間で各々、種々の variation を吸収する。例えば（問題－設計）という層では問題に対する種々の設計法をサポートする。このような階層的な記述

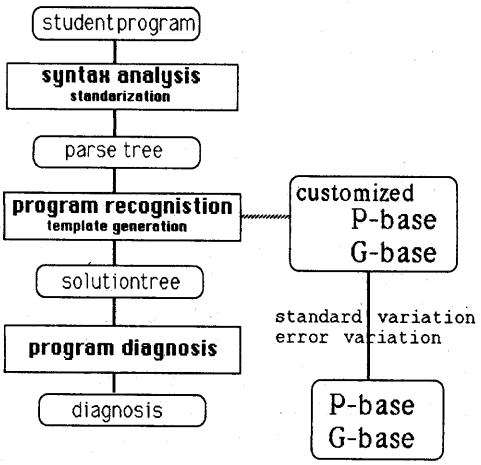

図 2: プログラム診断の流れ

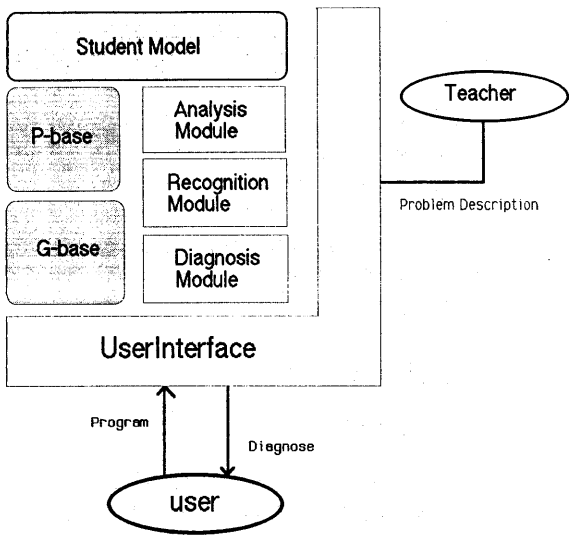

図 3: プログラム診断システムの概要

により一つの問題に対する種々の program の variation のサポートをコンパクトに可能としている。また誤ったプログラムに対しても、各層の間での誤解に基づく誤りとしての記述を可能とする。さらにより細かな誤りについては学生プログラムの標準化及びテンプレートの前処理時の自動誤り変換により対応している。

以下2章ではシステムの全体的な構成を、3章では問題に対する記述方法について、4章では階層的な記述だけでは扱いきれないバリエーションの取り扱いについて、5章ではプログラム理解に基づく診断についてそれぞれ述べる。

2 プログラム理解システム

PascalTutor はあらかじめ与えられた問題群に対して、(問題、プログラム) の対を入力として、その問題の解としてのプログラムを理解し、診断を行なう。問題に対してはその種々の解 (プログラム) が各々の段階的詳細化のステップを融合した様な形態で記憶されており、その中のどのパスを経由して設計されたと考えられるかの認識としてプログラム理解を行なう。そのような意味から個々の問題に対してその種々の解法の表現を一種の and-or 木として表現したものを問題解法木と、そして与えられたプログラムに基づいて問題解法木の or パスを特定したものを解法木と呼ぶ。**PascalTutor** によるプログラム診断のながれを図2に、システムの構成を図3にそれぞれ示す。

システムは UNIX 上で CommonLisp を用いて実現されている。大きく分けて解析モジュール、認識モジュール、診断モジュール、問題知識ベース、ユーザインターフェイス、そして学生モデルの6つからなる。

プログラムはまず解析モジュールで構文解析され、リスト形式の構文解析木に変換される。次に問題毎の問題知識ベース (問題解法木) とこの構文解析木とを照合し、プログラム理解を行ない、その結果としての解法木をつくる。次にこの解法木と教授知識に基づいてプログラム診断を行なう。

問題知識ベースは問題固有の知識ベース (Pベース：問題の設計法を記述したもの) と、問題独立の知識ベース (Gベース：種々のゴールの標準的な実現方法をゴール・プランとして記述したもの) からなる。この2つを分けることによりGベースのライブラリ化が計られ、またシステムの効率化も計られる。ある程度十分なGベースが用意できれば、問題の知識ベースへの登録はPベースへの登録だけで良いことになる。基本的にはシステム設計者がPベースへの登録を行ない、教師はGベースをプログラミング課題に応じて作成すればよい。Pベース、Gベースはそれぞれ問題単位、ゴール単位でモジュール化されており、プログラム理解においては必要な部分のみロードすればよい。

PascalTutor でのプログラム理解の基本動作はテンプレートとプログラム (片) とのパターン照合である。テンプレートは Pascal での予約語、記号とパターン変数と呼ぶ照合のための変数等から作られる。パターン変数は構文的属性を持っており、その属性と一致する任意のプログラム (片) と照合する。まだ一度も照合されていないパターン変数は未バイ

ンドであるといい、照合されることにより既バインドとなり、照合したプログラム片を値属性として持つ。既バインドのパターン変数はまた他のゴール（集合）の照合の対象となる。これにより階層的な問題記述を可能としている。

教授知識ベースはユーザのプログラム設計に対する誤りに基づく診断メッセージを持ち、また教授戦略の制御も行なう。解法木と学生モデルに基づき、個々の学生に合わせた診断と指導を行なう。誤ったプログラムであっても解法木さえ構成できれば、学生の設計の意図を推量して、その意図の上に立った診断を可能としている。

3 問題知識ベース

3.1 問題知識ベース

PascalTutor ではゴール間の依存関係を考慮した、goal-plan 構造による問題の記述法を採用している。PascalTutor によるプログラム理解は問題解法木と学生プログラムとのパターン照合を基本とするが、この方式においては次の様な基本的な問題がある。

1. 設計を構成する各ゴールの間には依存関係が存在する。例えばある設計のなかでゴールAを使用したならば、ゴールAのなかでゴールBを使用しなければならない等。この依存関係をどのように表現するか？

2. 一つの問題に対する variety は非常に多く存在するこれらをどの様な基準のもとに分類し、認識すればよいか？

PascalTutor では（1）に対してはテンプレートとプログラムの照合の結果、パターン変数にバインドされたプログラムリストを他のゴールとの照合に利用するという形式でゴール間の制約条件を表現する。このために以下で述べる階層的な問題分割法を採用している。これにより段階的詳細化とほぼ同様な形態で問題記述が可能となる。

（2）に関しては問題の記述を（問題ー設計ーゴーループランーテンプレート）という階層構造で表現することにより、設計の誤り、ゴールの実現の誤り等を区別することを可能としている。さらに

- 学生プログラムの標準化
- 標準誤り変換
- プログラムの等価規則

によって問題に対する設計の数、ゴールに対するプランの数、そしてプランに対するテンプレートの数の軽減を計っている。

PascalTutor での問題の記述は2章でも述べた様に、（問題ー設計ーゴーループランーテンプレート）という階層構造に基づく。この中心になるのはゴールである。

プログラム理解には認知心理学をベースとして人間によるプログラム理解をモデルとするアプローチもあるが、PascalTutor ではソフトウェア工学的な見地からプログラム理解をとらえ、プログラムの段階的詳細化のステップを再現するという形でプログラム理解を行なう。そこで問題となるのは段階的詳細化の各ステップにおける基本要素である。段階的詳細化では各ステップで基本要素を設定し、基本要素をベースとしてプログラムを設計する。そして次にその基本要素に対して同様の段階的詳細化を行なう。このような基本要素には種々のものが考えられる。あるレベルでは問題自体が基本要素になるし、また基本的な実行パターンも基本要素となる。

PascalTutor ではこの基本的な実行パターンを中心にとらえ、これをゴールと呼ぶ。例えば初心者の演習問題でよく現れるパターンとしては「入力データがある条件を満たすまである処理を繰り返すといったものがある。この様なゴールは抽象的な機能であり、これを実現する（Pascal による）プログラム（片）は種々存在する。たとえば

```
while 文による実現
  read(data)
  while data 条件 do begin
    処理
    read(data)
  end

repeat 文による実現 1
  repeat
    read(data)
    if data 条件 then
      処理
  until not data 条件

repeat 文による実現 2
  flag=FALSE
  repeat
    read(data)
    if data 条件 then
      処理
    else
      flag=TRUE
  until flag
```

これらの実現法をプランと呼ぶ。プランはゴールの特定の言語（PascalTutor では Pascal）による

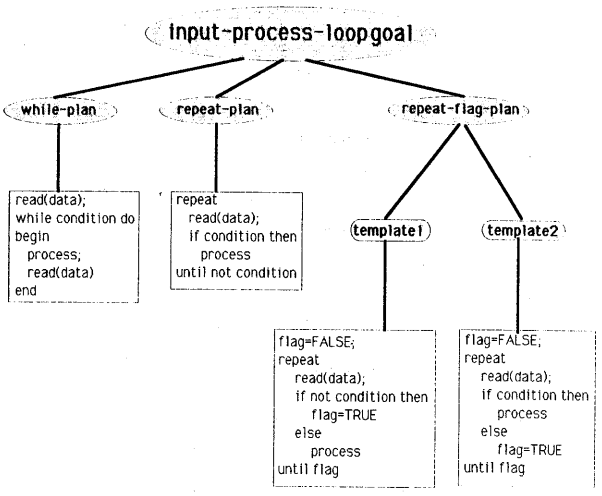

図 4: プランとテンプレートの variety

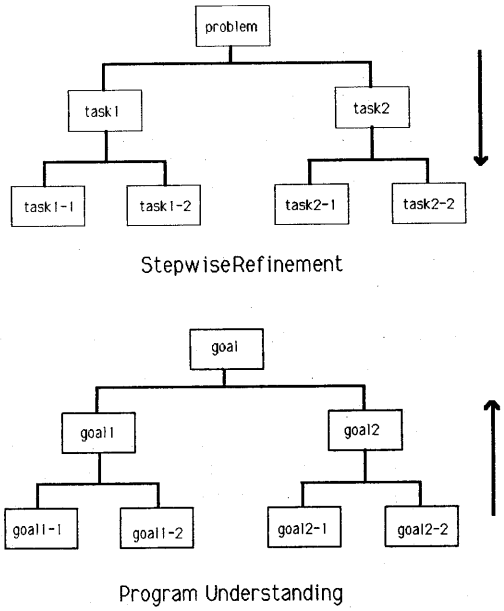

図 5: 段階的詳細化とプログラム理解

実現法を記述するものであるが、その実現法にも種々の細かな variation があり得る。この variation 毎の実際の実現をテンプレートと呼ぶ。大抵の場合はプランには一つのテンプレートが対応するが、プランによって2つ以上のテンプレートの対応するものもある。例えば上記の条件付き入力一処理ループの repeat 文による実現では終了条件の判定の仕方により2つのテンプレートが存在する。(図4参照)

問題の記述はこのゴールをベースとして、以下に述べる問題記述言語を利用して、時には既に解法の定義された他の問題を利用して、トップダウン的に行なう。問題の一つの設計をこのようにゴールをベースに構成することを問題のゴール分割と呼ぶ。段階的詳細化と PascalTutor でのプログラム理解の関係を図5に示す。段階的詳細化では図5の様にサブタスクへの分割をトップダウン的に繰り返すが、プログラム理解では最初のサブタスクに相当するゴールのテンプレートを照合することから始まる。そしてその照合が成功すれば、照合によって得られたサブコンポーネントと下部のゴールとの照合を行なって行く。ただしプログラム理解では最初から特定の設計に限定される訳ではないので、途中で失敗することもあり得る。その場合にはバックトラックを行ない、他のゴールを試みる。

3.2 Pベース

Pベースは図1に示した通り、ある問題に対してどのような設計が存在するかという問題記述と、ゴール(および定義済みの問題)を利用して設計がどのように行なわれるかという設計記述の2つの記述からなる。

PascalTutor で行なうプログラム理解はある方向に沿ったプログラムの理解であってブラックボックスとして、入出力仕様のみ一致する様なプログラムの認識ではない。真の意味でのプログラム理解は入出力仕様さえ一致すれば問題に対する正しい解として認識する必要があるが、考え得る(可能として)すべての variation を考慮する、または構成要素からその意味を推論するというアプローチは現在のところ実用的ではない。PascalTutor の目的とするところは初心者のプログラミング言語教育におけるプログラミング演習のCAIであり、そのような状況においては問題毎に採用すべき標準的な好ましいパラダイムが存在し、いくら入出力仕様が一致していてもそのようなパラダイムからはづれたものは正しい解として認識するする必要はないと考える(正しくない理由をシステムが把握することは必要ではあるが)。本方式では好ましいと考え得る設計方針に基づくプログラムのみを正しいと認識し、またその様なプログラムをベースとした誤りのみを認識する。そのため根本的に異なった設計思想に基づくプログラムは理解できない。

以下では次の様な問題を例とする。

『配列 data に ndata-1 個のデータが記憶されている。キー key を入力し、それが data に入っているかどうか判定せよ。入っていなければ ndata 番目のデー

```
(problem-frame
   'problem2
   "linear search and registration"
   '((variable "?data" "data")
     (variable "?count" "ndata-1")
     (variable "?key" "key"))
   '(design2-1 design2-2 design2-3)
   '(design2-4))
```

図 6: 問題記述の例

タとして登録し、ndata を更新せよ。負のデータが入力されるまで繰り返し上記の処理を行なえ。配列 data の下限は 1 とする。キーが data に入っている時は "key is found" と、入っていない時には "key is not found" と出力する。数 EOD を入力するまで数の列を読み込み、その平均を計算せよ』

問題記述では問題に固有なパターン変数（問題変数と呼ぶ）の宣言を行なった後、種々の設計をリストアップする。問題の実現の種々のバリエーションの中には設計の間違いと考えるのが最も適当であるプログラムも存在する。そこで問題記述では正しい設計記述とともに誤った設計記述もリストアップする。問題変数は問題の記述の中で明示的に使われる変数であり、どのような設計を行なうかに関わらず必ず使われる変数である。パターン変数はすべて構文の型を属性として持っており、属性の一致するプログラム要素とのみ照合できる。問題 1 に対する問題記述の例を図 6 に示す。図 6 では 3 つの問題変数に対して、その値を前もって定めている。この様にした場合には例えば問題変数?data は名前 data を持つ変数としか照合できない。

図 6 の記述では正しい設計記述を 3 つ、誤った設計記述を 1 つ宣言している。この問題記述自体を他の問題の設計記述で利用してもよい。その時には問題変数は仮パラメータ的な役割を果たす。

設計記述ではトップダウン的な問題の解法の分割と順次的な問題の分割を組み合わせて、あるパラダイムに添った設計を記述する（ゴール分割と呼ぶ）。また設計固有のパターン変数を設計変数として定義する。

ゴールの水平分割は図 7 に示す 5 つのゴールの組合せとして記述する。この内主ゴールは必須であるが、他のゴールは必要に応じて利用する。

主ゴールは更なる分割は行なわない。トップダウン的なプログラム理解ではまず何か設計上の最も主たる機能を実現している部分を探す。そしてそれをベースにして他の部分を認識し、主たる部分の認識を確実なものとする。PascalTutor では主ゴールがその役割を果たす。ゴールとは何らかのプログラムのパターンであり、プログラムとゴールとの照合により、プログラムは基本的には次の 5 つの部分に

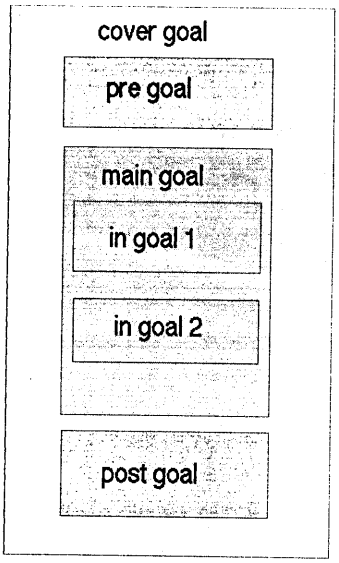

図 7: 5 種類のゴール

分割される。（図 8 参照）

- A．ゴールと照合された部分
- B．A の上にくる文の列
- C．A の下にくる文の列
- D．A の中にくる文の列
- E．A および B、C を囲む構文

B 部分はいわば A 部分の処理の前処理的な部分であり、前置ゴールと照合を行なう。C 部分は A 部分の後処理的な部分であり、後置ゴールと照合を行なう。D 部分は A 部分を段階的に詳細化した時の階層に相当し、A 部分の形態により任意個存在する。E 部分は全体を囲む制御部分であり、純粋な段階的詳細化では存在しないが、プログラム理解においては内部の処理をまづ第一に考えることもあり得るため必要となる。カバーゴールと照合する。設計記述の例を図 9 に示す。内部ゴールは主ゴールの照合でバインドされたパターン変数に対して対応付けられており、そのバインドされたプログラム片と照合される。

階層的な分割はパターン変数にバインドされたプログラム片の内容を指定する形で行なう。即ちバインドされたプログラム片を次の 3 つの形態のうちのどれかとして記述する。

- 水平分割されたゴールの組合せ
- 単一のゴール

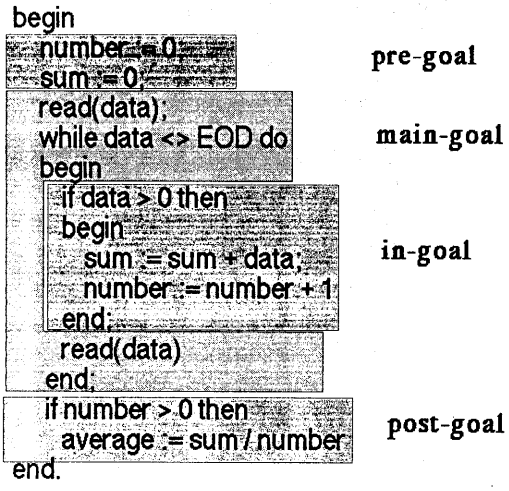

図 8: ゴールとプログラムとの対応

```
(design-frame
 'design2-1
 "linear search and registration"
 '(goal1)
 'problem2
 '((statement "?statement1")
   (statement "?statement2")
   (statement "?process"))
 '((main-goal (type goal)
              (name goal4)
              (apara "?key" "?key>=0" "?process"))
   (in-goal ((type goals)
             (object "?process")
             (goals
              (main-goal (type goal)
                         (name goal1)
                         (apara "?data" "?count"
                                "1" "?key"
                                "?statement1"
                                "?statement2"))
              (in-goal ((type goal)
                        (object "?statement1")
                        (name goal2)
                        (apara "'key is found'"))
                       ((type goal)
                        (object "?statement2")
                        (name goal3)
                        (apara "?count:=?count+1"
                         "'key is not found'"))))))))
```

図 9: 設計記述の例

```
(goal-frame 'goal1
            'statement
            "linear search using sentinel\\
             set the sentinel at the last element \\
             and search until sentinel/key is found"
            '((variable "@data")
              (variable "@count")
              (constant "@init")
              (variable "@key")
              (statement "@statement1")
              (statement "@statement2"))
            '(plan1-1 plan1-2)
            '(plan1-3))
```

図 10: ゴール記述の例

- 既定義の（別の）問題

図 9 ではまづ全体を main-goal と in-goal に分割し、次に in-goal に対応したプログラム片を更に main-goal と in-goal に分けている。

3.3 Gベース

Gベースは設計記述で使われるゴールとプログラムパターンたるテンプレートとの対応を記述するゴール記述と、テンプレートの実現法を pascal プログラムの形式で記述するプラン記述の 2 つからなる。PascalTutor では問題のゴール分割とゴールのプランによる実現をプログラミング教育の中心と考える。そこでGベースは標準的なゴールにはどのようなものがあるか、またそれらのゴールは pascal ではどのように実現すればよいかという観点から作成する。Pベースと Gベースを分離するという立場からゴール分割のベースとなる種々の標準的なゴールを多数用意することが必要となる。

ゴール記述では正しいプランと誤ったプランを記述する。ゴール記述においても固有のパターン変数が使われるが、それをゴール変数と呼ぶ。ゴール自体及びゴール変数は指定された構文に対してのみ照合する。ゴール変数はいわばゴールの仮引数であり、設計記述でのゴールの引用時の (apara ....) の部分が対応する実引数になる。例えば図 9 では goal1 をそれぞれ "?data","?count" 等で具体化したものでプログラムを作ることを示している。

図 10 では 2 つの正しいプラン記述（plan1-1 と plan1-2）と、1 つの誤ったプラン記述（plan1-3）を定義している。図 10 の goal1 は構文属性の値として statement を持つので、statement として認識されたプログラム片とのみ照合できる。

プラン記述ではゴールに対するパターンを pascal の文（型）として与える。具体的に pascal の構文が使われるのはこの部分であるので、言語毎のプラン記述を用意することにより、複数の言語に対してプラン記述の切替えだけでプログラム理解を行なうことも可能である。プラン記述においても固有のパター

```
(plan-frame 'plan1-1
            "linear search using sentinel\\
             set the sentinel at the last element \\
             and search until sentinel/key is found\\
             You used while statement for this goal"
            'goal1
            '((variable "@i"))
            "@data[@count]:=@key; @i:=@init;
             while @data[@i]<>@key do
                @i:=@i+1;
             if @i < @count then
                @statement1
             else
                @statement2"
            nil)
```

図 11: プラン記述の例

ン変数が使える。これをプラン変数と呼ぶ。テンプレートはゴール変数とこのプラン変数を利用して記述する。図１１にプラン記述の例を示す。この例では誤ったテンプレートは指定していない。

4 各種バリエーションの取り扱い

与えられた問題に対するプログラムには多くの variation が存在する。ことに誤ったプログラムまで含めれば数限りない variation が存在する。これらの variation は変換規則、誤り規則等によりできるだけ吸収するが、すべての可能性を考慮することは対話性の上からも難しい。そこで PascalTutor では詳細な理解に先だって key となるゴール（群）の認識によりプログラムが採用している設計法を特定するというアプローチを採用した。この方式により設計法から大きくはづれる誤った（又は不適当な）設計に対してはこのようなキーゴール（群）のみを与え、その認識のみで診断を行なうことが可能となる。図９の設計記述の例では５行目の '(goal1) がそのキーゴールである。

4.1 Ｐｌａｎ数の軽減のための変換規則

ゴールに対するプランはその具体的な実現法であるが、プランのなかでそのすべての variation をカバーすることは徒にプラン数の増大を招くだけであり、好ましくない。そこで PascalTutor では次の２つのアプローチをとる。

1. プログラムの標準化
   学生プログラムをある程度前もって標準化し、その上でプログラム理解を行なう。但し目的は診断であるから過度の標準化は元のプログラムの内容を損なうことになる。そこで PascalTutor では文単位での標準化のみを行なう。標準化の例としては次のようなものがある。
   - カッコの展開および項のまとめ
   - 定数の計算
   - 比較演算子の制約 <= は >= に、< は > に変換する。
2. プランのテンプレートの自動生成
   ある程度標準的な変形は異なったプランとは見なさないで、プランに変形規則を適用することにより対応する。標準的な変形としては次のようなものを導入している。
   - 特に制約されていない（交換可な）２項演算子でつながれたリストは交換してもよい。例えばテンプレート中の代入文の系列はその間に依存関係が無ければ、交換したものをその variation として用意する。全体としてフロー解析によりこのような変換を行なうには時間的余裕がないが、テンプレートという小さな単位だと可能である。
   - if 文の then 部分と else 部分の交換
   - write 文の排除

4.2 誤りの取り扱い

学生のプログラムには種々の誤りが存在する。PascalTutor ではそれらの誤りを以下の様に分類して認識する。

1. ゴール分割の誤り
   問題記述レベルの誤りであり、設計法が誤っていることになる。設計の誤りを前もって考察し、正しい設計記述に対する差分として、誤った設計を記述する。また前もってあるゴール（の集合）を設計に対応させて用意しておき、このゴール（集合）の部分的な照合により誤った設計を判断する。例えば前記の例題において for 文を使っておれば、データの入力による繰り返しという機能の実現方法を基本的に誤っていると考える。
2. ゴールの実現法の誤り
   ゴールに対して要求されている機能を誤解して、誤ったテンプレートを対応させていると考えられる誤りで、ゴールに対する誤ったプラン記述として記述する。
3. 局所的にとらえ得る誤り
   類似の演算子の使用等設計法とかゴールとかに関係なくとらえ得る誤りで、プランに対する誤ったテンプレートとして与える。但しＧベースの作成者が作成するというアプローチはとらず、Ｇベースの前処理によりシステムが個々のプラン記述に対して自動的に作成するというアプローチを採用している。標準的な誤り変換規則として以下の形式でデータベース化し、テンプレー

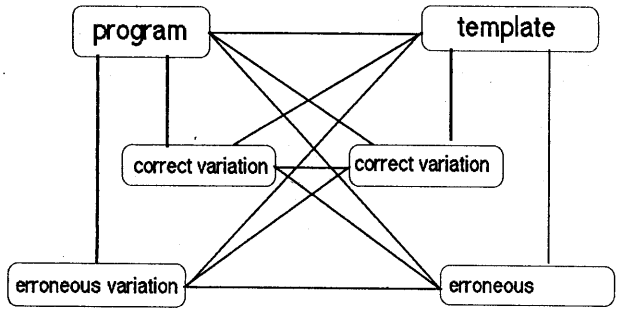

図 12: プログラム片とテンプレートとの照合

トがこの条件を満たせば照合時に誤りテンプレートを自動生成し、これとの照合を試みる。

IF テンプレート条件 THEN 変換規則

実際のプログラム片とテンプレートとの照合は図１２の順で行なうことになる。

5 プログラム診断

システムはプログラム理解の結果に基づいてプログラム診断を行なう。PascalTutor におけるプログラム理解とは学生がプログラムを作成するに当たって、どのような設計法に基づいて、どのようなプランを用いたかの認識であり、誤ったプログラムについても設計法が誤っているのか、それともゴールに対するプランが誤っているのかといった認識になる。この様な認識の結果が解法木として表現されている。診断の目的はこのような誤りを学生に理解させることであるため、診断としては単にテンプレートが間違っているといったメッセージではなく、全体の設計のうえからの必然性を指摘した上で誤りを指摘する。

図１３はゴールに対する誤ったプランを使っている場合の診断の例である。PascalTutor では基本的にトップダウン的な診断とボトムアップ的な診断の２つを組み合わせて全体の診断を行なう。

トップダウン的な診断とはゴール分割の誤っているゴールを設計記述の上での意味から説明するもの

```
010 program problem2(input,output);
020 var key,ndata,i:integer;
030     data:array[1..100] of integer;
040     begin
040       read(key);
050       while key>=0 do
060       begin
070         data[ndata]:=key;
080         i:=1;
090         while data[i]<>key do
100           i:=i+1;
110         if i<ndata then
120         begin
130           ndata:=ndata+1;
140           writeln('key is not found')
150         end
160         else
170           writeln('key is found');
180         read(key)
190       end
200     end.

DIAGNOSIS
Your design strategy for this problem is as follow:

  ....................................

You misunderstood "the search index usages".
You must do the following task:

  if the search index is greater than or equal to
     the final index
  then
     key is not found
     increment the final index
  else
     key is found

but you did as follow:

  if the search index is lower than the final index then
     key is not found
     increment the final index
  else
     key is found

So your mistake is

     i<ndata

and you must correct it as follow:

     i>=ndata
```

図 13: 診断例

で、いわば問題に依存した診断ということになる。図１３では前半の診断がこれに当たる。この診断はＰベースに基づいて行なう。ボトムアップ的な診断とはゴールに対してプランの実現がどのように誤っているかを説明するもので、いわば問題に独立な診断ということになる。図１３では後半の診断がこれに当たる。この診断はＧベースに基づいて行なう。この様に診断を２つのレベルに分けることにより、とりあえずどちらか一方のみの診断を行なってユーザの反応をみて、必要に応じてさらなる診断を行なうことも可能であるし、同一のゴールに対する実現の誤りを繰り返し行なっているといったことを判定して、その部分に対してより詳しい診断を行なって、場合によってはドリルを行なわせることも可能となる。

6 結び

本論文では（問題－設計－ゴール－プラン－テンプレート）という階層性に基づいた問題の記述と、それによるプログラム診断システムについて述べた。PascalTutor は段階的詳細化によるプログラム設計を辿る形でプログラム理解を行ない、設計法およびゴールの実現法の両面からの診断を行なう。また標準的な変換および誤り変換により問題に対する知識ベースをむやみと大きくすることなく種々のプログラムの variation に対応することを可能としている。

プログラム理解はあらかじめ定義された設計法にもとづいたプログラムしか認識できないという欠点はあるが、標準的な問題の設計法を教えるという観点からは問題ないと考える。

現在Ｐ－ベースは５つの問題について実現しており、Ｇ－ベースには約５０個のゴールを登録している。学生の実際のプログラム（約５０例）を検討した範囲でもこれでは十分ではなく、すべてを認識するにはさらに多くのプランの variation を必要とする。しかし設計のゴール集合による前照合を使った認識により標準的な設計法から大きくはづれているものは除くことができる。そうすることにより比較的小数のＧ－ベースにより認識が可能となるのではないかと考えている。

参考文献

[1] A.Adam J.Laurent:LAURA,A System to Debug Student Programs, Aritificial Intelligence 15 (1980)

[2] W.Lewis Johnson,etc: PROUST:Knowledge-Based Program Understanding,ICSE (1984)

[3] B.J.Reiser,etc.:Dynamic Student Modelling in an Intelligent Tutor for Lisp Programming,Proc. IJCAI (1985)

[4] 上野: アルゴリズム知識に基づくプログラム理解の枠組み, 人工知能学会研究会資料 (1989)

[5] William R.Murray:Automatic Program Debugging for Intelligent Tutoring Systems, Pitman, Readings (1988)