コンピュータと教育　10－14
(1990. 4.25)

プログラム理解に基づくプログラミングチュータ

Programming Tutor Based on Program Understanding

海尻　賢二

Kenji Kaijiri

信州大学工学部情報工学科

Shinshu University,Faculty of Engineering, Information Engineering

あらまし　プログラム開発は高度に知的な作業であり、それをマスターするために長い知的訓練を必要とする。そのためプログラム開発の種々の側面を支援するプログラミング環境の研究は幅広く行われている。しかしながら多くのシステムは現実のプログラム開発作業を支援することを目的としており、初心者のサポートを目的とはしていない。そこで初心者のプログラミング教育の種々の側面をサポートすることを目的としたプログラミング教育用知的ＣＡＩシステムの実現上の諸問題について論じ、現在実現中のプログラム理解に基づくPascalプログラム診断システム(PascalTutor)について紹介する。

キーワード　情報処理教育　ＣＡＩ　知識ベース　知識工学

1。序論

プログラミング教育は非常に幅広い側面を持ち、計算機による援用が非常に有効に働く分野である。しかし現在の計算機システム（環境）は初心者の教育という観点からから見ると非常に貧しいサポートしか行っていない。そのため初心者はプログラミング教育の種々の側面の理解のために本質的でない多くの付加的な作業に悩まされることになる。

そこで本研究では初心者のプログラミング教育をトータルに援用する環境（Intelligent Programming Tutoring Environment--IPTE）について考察する。この環境では学生プログラムの理解をその中心におき、チュータという立場で支援を行う。本論文ではＩＰＴＥの全体の構成及びその中心となるプログラム理解について、そしてプロトタイプとして実現したＰａｓｃａｌに対するＩＰＴＥ（ＰａｓｃａｌＴｕｔｏｒ）について述べる。

以下２章ではＩＰＴＥ全体の概念を、３、４、５章ではＩＰＴＥのコンポーネントとして実現した種々のシステムについて、そして６章ではシステムの中心となる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。プログラミングチュータ

プログラミング言語およびプログラミングの教育はその重要性をますます高めつつあるが、既存の計算機の、そのための援用環境は非常に貧しいといわざるを得ない。コンパイラーつをとってみても誤りメッセージは（ＷＳやメインフレームでは）英語であるし、またその内容も誤りの内容を初心者向けに的確に表現したものとはいえない。そしてコンパイラ以外については全く援用していないに等しい。そのため初心者は本来学ぶべき内容以外のことの理解に多大の時間を費やされることになる。もちろんこのような内容の理解も最終的には必要であるが、例えばプログラミング言語の学習において個々の誤りメッセージの内容の理解は当面必要ではない。

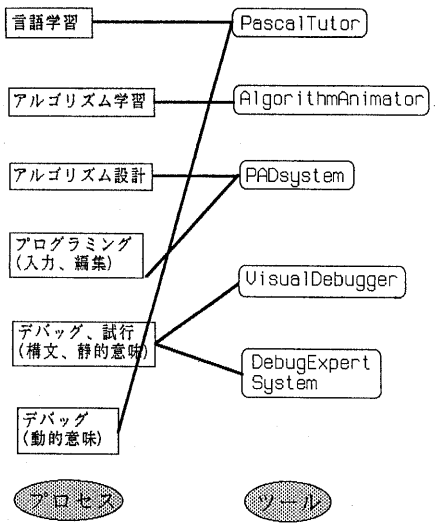

図１　知的プログラミング教育支援環境

そこでプログラミングを学んでいく際、その時点時点で学ぶべき内容に焦点を合わせて、それ以外の内容をできるだけシステムが援用し、初心者の負担を軽減することを目指して、初心者用プログラミング教育援用環境（Intelligent Programming Tutoring environment -- IPTE）を実現中である。システムの概要を図１に示す。

プログラミング教育はソフトウェアのライフサイクルで言うと、設計、実現、保守のプロセスを教えるものであり、具体的には以下の様な種々の側面からなる。
（１）プログラミング言語の学習
（２）デバッグ手法の学習
（３）計算機システムの使い方の学習
（４）プログラム設計法の学習
（５）アルゴリズム設計法の学習
（６）種々のアルゴリズムの理解
（７）ドキュメントの作り方の学習

ＩＰＴＥではこれらの学習を支援する為に次の様なサブシステムを考えている。
（１）プログラミング言語チュータリングシステム
（２）デバッグエキスパートシステム
（３）ｐａｄシステム
（４）アルゴリズムアニメーションシステム
（５）ビジュアルデバッガ
（６）プログラム診断システム

各サブシステムは現在は別々に作成を行っているが、将来的には各種データを（知識）データベースとした統合知的プログラミングチュータリング環境とする予定である。そして全体をプログラミング演習の支援という立場から捉える。

初心者のプログラミング教育という立場からみるとプログラミング演習は次のようなプロセスからなると考えられる。
（１）対応する言語概念の教授
（２）アルゴリズムの説明
（３）アルゴリズム、プログラムの設計
（４）プログラム入力
（５）試行、検査
（６）デバッグ
（７）文書化

ＩＰＴＥはこれらのプロセスを援用する訳であるが、実際には２種類の援用が存在する。一つはそのプロセスの学習の援用であり、一つはプロセス自体の援用である。例えばデバッグというプロセスにおいては、デバッグそのものをどのようにして行うかということの教授と、できるだけデバッグの負担を軽減して、人間がデバッグに要する労力をできるだけ計算機で肩代りするという意味での援用がある。もちろん後者の立場で援用しながら、そのメッセージとしての誤りメッセージ（誤り現象）からその原因を見つけるプロセスを教えていくという形態もあり得るが、基本的には別々に捉えるべきである。ＩＰＴＥでは後者の立場を中心としながら一部前者の立場を取り入れる。

3。プログラミング言語チュータリングシステム(1)

プログラミング言語の学習は構文、意味の説明を行い、次にその例題、そして最後に種々の演習というステップを経て行われる。プログラミング言語の種々の概念は互いに関連性を持っており、学習の各時点で関連する知識の確認が必要となる。また演習においては単に言語の知識のみならず、アルゴリズムの知識、問題領域のいわゆるドメイン知識、そして計算機アーキテクチュアの知識などが必要となる。

学生は学習の途中で別の単元の内容や、現単元中の他の項目を参照する。この参照はあくまでも一時的なものであり、参照後は元の学習に復帰する。このような構成の為には HyperTextが最も適している。そこでHyprTextを部分的に実現したシステムであるHyperCardを利用してPascalのチュータを試作した。システムの一画面を図２に示す。

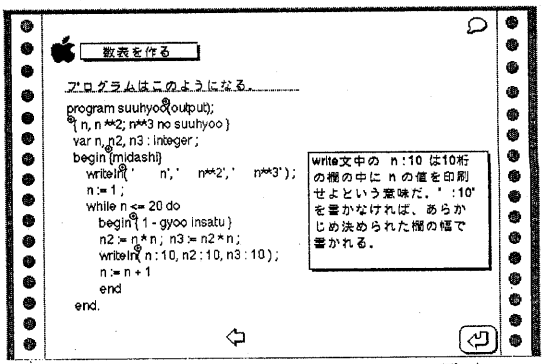

図２　ＰａｓｃａｌＴｕｔｏｒの一画面

全体は各単元を一つのノードとするネットワークで構成する。また各単元の学習は解説、例題、問題、演習の順に行う。解説は構文図に基づく構文の説明と意味の説明からなり、各々の項目中の用語は各単元のその項目の説明とリンク（HyperCardではあるエントリとあるカードを対応付けるためにそのエントリからカードへリンクを張る）が張られている。単元の順序関係を表すカードを図３に、単元内の項目の順序関係を表すカードを図４にそれぞれ示す。

システムはPascalの言語の知識をネットワークとして表現した教材知識ベース、この知識ベースに対する

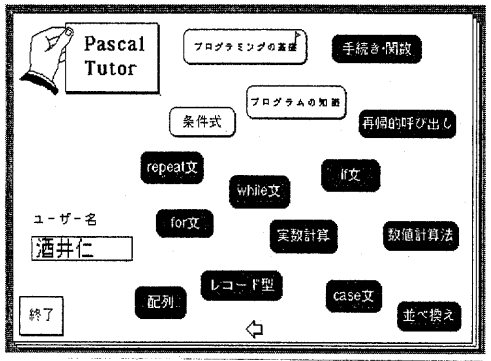

図３　単元の順序関係

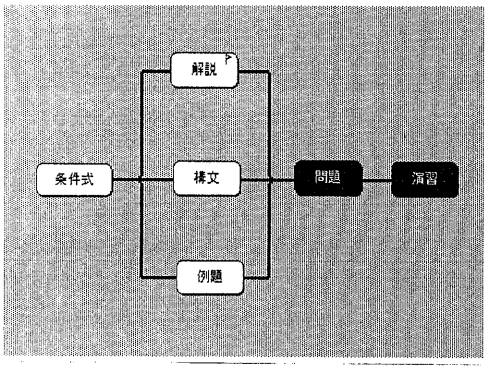

図４　単元内の項目の順序関係

オーバレイモデルとして表現した学生モデル、そして全体の制御を行う教授戦略からなる。ユーザインターフェイスはHyperCardのもつユーザインターフェイス機能をそのまま利用している。教材は順序関係を持ち、その順序関係をはづれないかぎり、ユーザは単元を自由に選択することができる。

一つの単元のなかでは解説、例題、問題、演習の順が指定されており、この順で学習を行う。『問題』は選択問題、記述問題、穴埋め問題の３つからなる。『演習』は与えられた課題に対してその場でプログラムを作成し、システムがその診断を行う。後述するプログラム診断システムはこの目的に利用する。これは現在は独立したシステムとして試作中である。

HyperCardの優れたＵＩ機能を使うことにより各単元、各概念の関連付けが容易に行え、またプログラムアニメーション等の機能を付加して例題のＵＩを向上させることが容易に行える。

4。デバッギングエキスパートシステム

デバッギングエキスパートシステムには次の２つの観点がある。

・デバッギングの方法を教えるエキスパートシステム
　デバッグそのものを助けるのではなく、デバッグというプロセスをどのように実現すれば良いかを教える。むしろデバッギングチュータとでもいうべきもの。
・デバッギングを助けるエキスパートシステム
　デバッギングを助け、ユーザのデバッグに費やす労力をできるだけ軽減しようとするもの。

従来行われているデバッギングエキスパートシステムは後者の観点からのシステムである。教育用という観点からは前者のシステムも重要であるが、今回は後者の観点からのシステムを試作した。

PascalDebuggerはプログラムとコンパイラの誤りメッセージを入力とし、その誤りメッセージをどのように解釈し、どのような訂正を行えば良いかを対話的に示唆する。現在はプログラムはソースとして利用していだけで解析は全く行っていない。エラーメッセージと解析情報に基づく支援が必要である。

5。アルゴリズムアニメーションシステム

プログラミング演習は初期の頃はともかく、ある程度進んで来ると種々の標準的なアルゴリズムの実現を通しての演習という形態をとる。この場合学生はプログラミングに先だって対象となるアルゴリズムを理解することが要求される。

アルゴリズムの解説は一般的にそのベースとなる論理的なデータ構造に基づいて行われ、場合によってそのデータ構造上での疑似シミュレーションという形で動作を教授する。複雑なアルゴリズムの場合このシミュレーションという教授形態が非常に有用である。

そこで何らかのアルゴリズム記述からアニメーションプログラムを（半）自動的に生成するシステムの試作を行っている。従来このようなアルゴリズムアニメーションは個々のアルゴリズムをどのようにアニメーション化すればよいかという観点から研究が行われており[2]、一般的にアルゴリズム記述からアニメーションプログラムをどのように作ればよいかという研究はほとんどない。

AlgorithmAnimatorではアルゴリズムをSmalltalkで記述する。Smalltalkはオブジェクト試行、継承というアニメーションシステムで有用な性質を持っている。AlgorithmAnimatorは次のような方法（手順）でアルゴリズムアニメーションを実現する（図５参照）。

（１）Smalltalkによるアルゴリズムの記述
（２）アルゴリズムの対象となる基本データ型をアニメ化されたものに変更する
（３）特に焦点を当てたい処理があればアニメ化された特別のメソッドで置き換える

（4）プログラムを実行する。

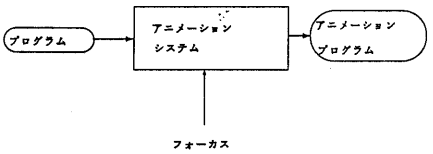

図５　AlgorithmAnimator

この方式であれば基本的なデータ型に対してそのアニメ化されたデータ型を用意しておけば、Smalltalkによるアルゴリズム記述の少しの変更によりアニメーションプログラムを実現することができる。

6。プログラム理解に基づくプログラム診断システム

ＩＰＴＥでは種々の側面からプログラムのデバッグプロセス（大きく言うと開発プロセス）をサポートする。前述のPascalDebuggerやalgorithmanimatorもそのためのサブシステムといえる。それ以外の重要な要素に論理エラー（現象としては実行時のエラーまたは予想外の出力）がある。これに対するサポートとしては次の２つを考えている。

一つはビジュアルデバッガであり、プログラムのビジュアルな実行シミュレーションを通してプログラムの理解を助ける。他の一つが本節で述べるプログラム理解に基づくプログラム診断システムである。これは与えられた問題に対するプログラムとして学生のプログラムを理解し、それに基づき論理エラーを含めての誤り診断を行う。(3)(4)(5)(6)

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の、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６に示す。

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（図７）。

プログラムはまづ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。

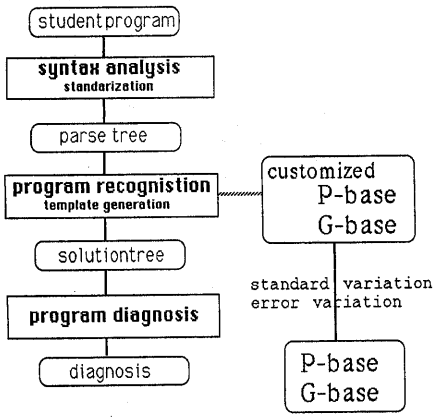

図６　PascalTutorによるプログラム診断の流れ

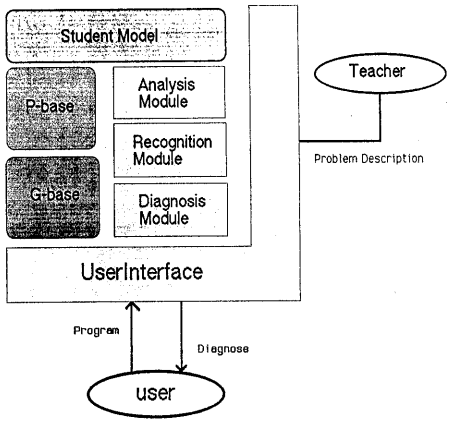

図７　ＰａｓｃａｌＴｕｔｏｒ

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）の２つからなる。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。

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。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。ＰａｓｃａｌＴｕｔｏｒにおける階層的なプログラム診断のフローを図８に示す。

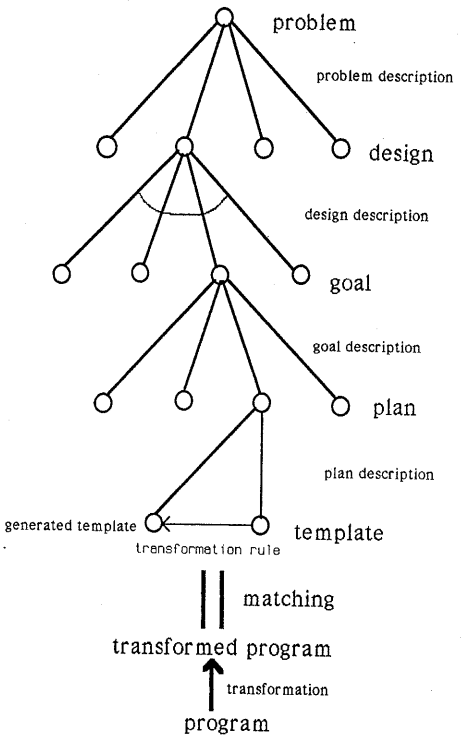

図８　階層的プログラム理解

6．1．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－設計－ゴール－プラン－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。このことにより多くのvariationを上記の規則の組合せとして表現することが可能となり、知識ベースの一般化を図っている。

PascalTutorでの問題の記述の中心になるのはゴールである。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。

問題の記述はこのゴールをベースとして、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。

簡単な問題に対するＰベースとＧベースの記述例を図９から図１２に示す。

６。２。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（現在システムの診断部の実現が完成していないので、この例では単に間違いの箇所を指摘しているだけである）。

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で、いわば問題に依存した診断ということになる。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。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。

７。結び

本論文では初心者のプログラミング教育という観点から、そのプログラム開発過程を総合的に、かつ知的に援用することを目的としてintelligent programming tutoring environmentを提案し、そのプロトタイプとして実現中の各サブシステムについて、特にその中心となる（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を主として述べた。この理解システム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少数の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。

現在はまだサブシステム毎に実現している段階であるが、将来的にはこれらを統合し、現実のプログラミ

ング演習の場でその効果を評価したい。

参考文献

[1]酒井、海尻：HyperCardに基づくプログラミング言語用知的ＣＡＩシステム、情報処理学会「教育におけるコンピュータ利用の新しい方法シンポジウム」、1989,12

[2]Marc H.Brown:Exploring Algorithms Using Balsa-II,IEEE Computer Vo.21 No.5 (May 1988)

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

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

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

[6]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)


```
(problem-frame
 'problem1
 "数の列をEODが入力されるまで読み込み,それらの数の平均値を計\
  算せよ。入力には負の数も含まれるものとし,平均には負の数及\
  びEODを含めないものとする。"
 '((variable "?data" "data" "個々の入力データを記憶する変数")
   (constant "?sentinel" "EOD" "番兵")
   (variable "?average" nil "平均値を代入する変数"))
 '(design1-1 design1-2 design1-3)
 '(design1-5))
```

図９ 問題記述の例

```
(design-frame
 'design1-1
 "番兵を使わないで入力と処理を繰り返す"
 'problem1
 '((goal4) nil)
 '((statement "?statement")
   (variable "?number" "データの個数をカウントする変数（カウン\
     トには番兵を含めない）")
   (variable "?sum" "データの和を代入する変数"))
 '((main-goal (type goal)
              (name goal4)
              (apara "?data" "?data<>?sentinel" "?statement"))
   (in-goal ((type goal)
             (name goal5)
             (object "?statement")
             (apara "?data" "?sum" "?number")))
   (pre-goal (type goal)
             (name goal31)
             (apara "?number" "0" "?sum" "0"))
   (post-goal (type goal)
              (name goal6)
              (apara "?number" "?sum" "?average"))
   ))
```

図１０ 設計記述の例

```
(goal-frame 'goal4
            'statement
            "番兵を除いた入力・処理の繰り返し"
            '((variable "@data")
              (expression "@condition")
              (statement "@process"))
            '(plan4-1 plan4-2)
            '(plan4-5 plan4-6 plan4-7 plan4-8))
```

図１１ ゴール記述の例

```
(plan-frame 'plan4-1
            "(1) read a data (2)repeat read-process using a \
           while statement;"
            'goal4
            nil
            "read(@data); while @condition do begin @process;
           read(@data) end"
            nil)
;
(plan-frame 'plan4-2
            "repeat read-process using a repeat statement"
            'goal4          nil
            "repeat read(@data); if @condition then @process
           until not @condition"
            nil)
;
(plan-frame 'plan4-5
            "(1) read a data (2)repeat read-process using a\
           while statement;you missed the first read statement"
            'goal4
            nil
            "while @condition do begin @process; read(@data) end"
            nil)
;
(plan-frame 'plan4-6
            "２つのread文の内の後半のread文を忘れている"
            'goal4
            nil
            "read(@data); while @condition do begin @process end"
            nil)
;
(plan-frame 'plan4-7
            "repeat read-process using a repeat statement"
            'goal4          nil
            "read (@data); repeat if @condition then @process;
           read(@data) until not @condition"
            nil)
;
(plan-frame 'plan4-8
            "you missed first read statement"
            'goal4          nil
            "while @condition do begin read(@data); @process end"
            nil)
```

図１２ プラン記述の例

```
>(prog-diag)
<<PascalTutorへようこそ>>
<<貴方のプログラムを調べます>>
<<貴方のプログラムは次のようなものです>>:

program average(input, output);
const EOD = 99999;
var
   data, s, a: real;
   x: integer;
begin
   x := 1;
   s := 0;
   read(data);
   while data <> EOD do begin
         if data > 0 then begin
            s := s + data;
            x := x + 1
         end
   end;
   a := s / x;
   writeln(a)
end. { average }


<<問題は次のようなものです>>:

数の列をEODが入力されるまで読み込み,それらの数の平均値を計
算せよ。入力には負の数も含まれるものとし,平均には負の数及
びEODを含めないものとする。

<<貴方は次のような変数を使っています>>:

変数"a"を"平均値を代入する変数"として使っている
変数"data"を"個々の入力データを記憶する変数"として使っている

<<貴方は次のような解法を使っています>>:

番兵を使わないで入力と処理を繰り返す

<<この解法のなかで次のような変数を使っています>>:

変数"s"を"データの和を代入する変数"として使っている
変数"x"を"データの個数をカウントする変数（カウントには番兵を
      含めない）"として使っている

<<貴方の使っている解法は次のような構造をしています>>:

 2つの変数を初期化する
番兵を除いた入力・処理の繰り返し
    条件付きの和
条件付きの平均

 <<貴方は次の誤りを犯しています>>

 貴方は
  ("条件付きの平均")
の実現の為にプログラムに対して次の誤り変換を適用した。
  ("writeln文を省く")

 貴方は
  ("条件付きの平均")
の実現の為にテンプレートに対して次の誤り変換を適用した。
  ("if-then文を省く")

 貴方は
  ("番兵を除いた入力・処理の繰り返し")
の実現の為に次の誤ったプランを使っている。
 "2つのread文の内の後半のread文を忘れている"

 (NIL NIL)

 >
```

図１３ プログラム診断の例

盛光印刷所