論　文

ゴール/プランに基づく初心者プログラムの認識システム

正　員　海尻　賢二†

Novice Program Recognition/Diagnosis System Based on Goal/Plan Concepts

Kenji KAIJIRI†, *Member*

あらまし　Reverse engineeringに，デバッグにと種々の観点からプログラム理解／認識の研究が行われている．ところで認識の範囲また種類は当然その目的とする所によって異なる．本論文では初心者のプログラミング教育の援用のためのデバッグ作業の援用という観点からプログラム認識をとらえ，そのための問題の記述法/認識システムの方式を提案し，プロトタイプシステムにより認識能力を評価する．本方式の知識ベースの記述の基本はゴール/プランであり，また認識の基本動作はパターン照合である．以下の特徴をもつ．

- 問題の解法を下降型でアルゴリズムレベルで問題解法木として記述する．
- 問題解法木はその意味として，構成要素たるゴール相互の位置関係の記述を含む．
- ベースとなる照合に種々のvarietyをもたせ，認識のレベルにより使い分ける．単なるパターン照合ではなく，テンプレートがプログラムの中で構造的に分散していても記述，認識できるようにした．
- 広範な誤りをできるだけ診断できるようにするために，下降型記述の特徴を生かし部分的な認識でも予測的に診断できるようにした．

なお実現したプロトタイプでは，初心者の良形でないプログラムサンプルに対して約85％の認識率を得た．

キーワード　プログラム理解，デバッグ，プラン，プログラミング教育，プログラミング環境

1. まえがき

プログラム認識は大きく分けるとreverse engineeringとデバッグという二つの観点から研究が行われている．

Reverse engineeringの立場からは[4),(10)]，(1)設計における意思決定を演えきし，再利用，保守に役立たせようというもの，(2)部品化プログラミングの立場から部品の候補を自動的に抽出しようとするもの，等があるが，基本となる前提としては(1)プログラムは論理的に正しい，(2)プログラムはある程度熟練したプログラマが作成しており，well-formedなものである，の2点がある．

それに対してデバッグの立場から[1)~(3),(5),(6),(8),(9)]，(1)初心者がプログラミングを学んでいくプロセスの援用の一環として役立たせようとするもの，(2)通常使われているシンボリックデバッガの更に高級なものとして一般プログラマに役立たせようとするもの，の2種類がある．そして基本となる前提としては，プログラムは何らかの誤りを含むものである，ということがある．

本論文では初心者に対するプログラミング演習におけるデバッグの援用を目的としたプログラム認識/ 診断システムPascalRecognizerについて述べる．

プログラミング演習は文法，アルゴリズムの双方の知識を必要とする．そのため初心者にとって意味的なバグのデバッグは両方の知識を要求されるため難しいものとなる．すなわちある意味的な誤りがあった場合，その原因としては使っている構文に対する誤解とアルゴリズム自体の誤りの二つの場合があり，デバッグのためには両方の知識が要求される．

そこで問題に即して意味的な診断を行うシステムを考える．そのため問題のいくつかの標準的な解法を知識ベースとして記述しておき，プログラムとそれとの相違により意味的なバグを認識/診断する．

† 信州大学工学部，長野市
Faculty of Engineering, Shinshu University, Nagano-shi,380 Japan

一般にこのような解法の記述はゴール/プランを使って行われる．ゴールとは問題に対する部分問題のようなものであり，プランとはゴールに対する種々の標準的な実現法である．問題の記述には認識に対応して上昇型の記述と下降型の記述の2種類がある．上昇型の記述（認識）では構成要素の記述が先にあり，アルゴリズムとしてではなくその組合せとして全体の記述を行う．そのためゴールの関係を厳密にはとらえないのでプログラムに対する融通性には勝るが，初心者のプログラム診断に必要な厳密性には劣る．それに対して下降型の記述（認識）では下降型でプログラムを設計する際の段階的詳細化プロセスとして問題をアルゴリズムとして記述する．そのため構成要素であるゴールの上位における位置関係を厳密に構造的に記述することができ，プログラムに対する融通性には劣るが，認識の厳密性には優れている．

従来この目的のための代表的なシステムとしてPROUST[2],[8], ALPUS[6]等がある．PROUSTでは認識という観点を重点的にとらえ，問題の解法を(問題の記述，ゴールの記述，プランの記述）という階層で記述する．しかし上昇的な認識を考えているために，問題記述レベルでのゴールの構造的関連についての記述は不十分であり，初心者のバグの多いプログラムの厳密な認識/診断は難しい．例えば同一のゴールが2箇所で使われていた場合，記述においてどちらを意図しているかはあいまいである．ALPUSでは階層的手続きグラフとして下降型の記述を行うが，構成要素の階層的な関係を記述するのみで，厳密な相互関係は記述しない．

PascalRecognizerでは設計（特に下降型設計）という観点から問題の解法を下降型の，ゴールの構造的な組合せとして記述する．これによりPROUSTではあいまいであった問題の解法における各ゴールの位置付けが明確となる．認識においても下降的に（予測的に）認識を行うので，全く認識できない部分に対してもあるべきゴールを予測して示唆することが可能となる．この点は初心者のプログラムを対象とする場合には特に重要となる．すなわち一部は理解してほぼ正しく実現しているが，他の部分は全く理解できていないようなプログラムであっても下降型であれば部分的な認識/診断が可能となる．そこでPascalRecognizerでは初心者のプログラムを詳しく診断するという目的から従来あまりとられていない下降型の記述法を考え，その認識法を実現した．PROUSTやALPUSにおいては認知科学的な概念を認識の中心においているが，リバースエンジニアリングの観点からのプログラム認識では解析的手法が中心で，認知科学的な概念はあまり用いられていない．初心者プログラムを対象とする場合，確かに認知科学的概念も必要にはなるが[11]，それは診断において主に重要と考える．本論文では認知科学的な概念をベースとはせずに，解析的手法でどこまで認識できるかを試みている．

PascalRecognizerではプログラミング教育という観点から問題に対して教えたアルゴリズムを正しく理解し，プログラミング言語で実現しているかを調べるという立場から認識のベースに抽象アルゴリズムというものをおき，これをプログラミングノウハウというべきプランに基づいて問題解法木と呼ぶ一種のand-or木で記述する．そして問題解法木とプログラムとの照合がとれるかをパターン照合をベースに判定する．特に広範な誤りを認識するために部分照合，部分認識という柔軟性のある戦略を採用している．そしてプロトタイプシステムではPascalを教えて数か月の学生に出した課題に対して約85％の認識率が得られた．本論文ではPascalRecognizerの知識ベースの記述と認識法を中心に述べる．以下**2.**では概要を，**3.**では知識ベースの記述法を，**4.**,**5.**では認識法について述べる．更に**6.**では診断の問題点について，そして**7.**ではシステムの評価を行う．

2. PascalRecognizer

PascalRecognizerはプログラム診断において図1に示すように解析・認識・診断というステップを経る．システムの全体構成は図2に示すようである．

- 解析モジュール

プログラムを構文解析し，リスト形式の構文解析木に変換する．更に標準化，簡単化も行う．

- 認識モジュール

下降型，段階的なパターン照合（プログラム断片とテンプレート）をベースとして，与えられた問題の解としてプログラムを認識し，解法木という形式にまとめる．

- 診断モジュール

解法木に基づいて学生の犯している誤りを識別し，適切な診断を行う．

- 知識ベースモジュール

問題ごとにその解となるプログラムおよびその誤りのバリエーションをデータベース化した知識ベースと，

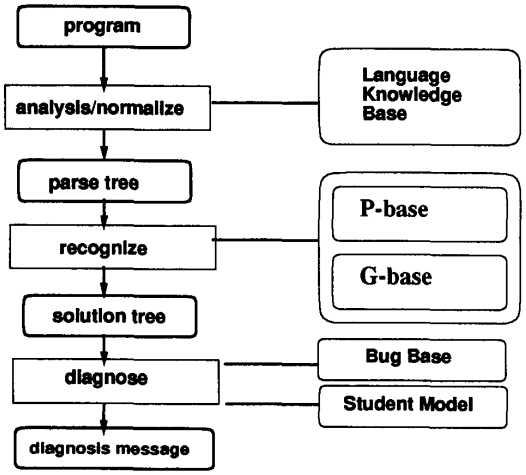

図1　プログラム認識の流れ
Fig. 1　The flow of program recognition.

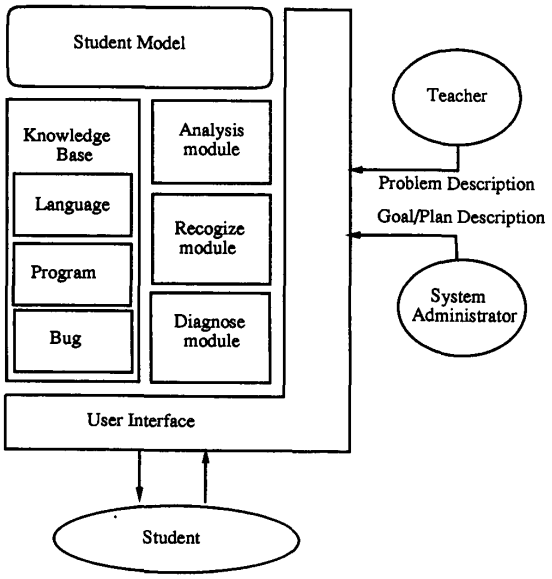

図2　PascalRecognizerの構成
Fig. 2　The PascalRecognizer.

その操作関数群である．

PascalRecognizerのユーザは，プログラムを与えて診断を受ける学生，課題の問題記述を与える教師，問題記述のベースとなる知識ベースを構築するシステム管理者の3者からなる．システムとしてはこの3者に対してインタフェースをもつ必要があるが，現在は学生に対するインタフェースしかもっていない．

3. 知識ベース

3.1 問題の記述法

PascalRecognizer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，その問題の解としてプログラムを認識し，診断を行う．PascalRecognizerでは問題の解法は下降型のゴールの構造的な組合せで表現できると考える．また初心者の場合，プログラム設計も段階的詳細化に基づいて行うとみなす．PROUSTでは問題を部分問題であるゴールに分割し，そのゴールをプランで実現すると考える．但しゴールの組合せは基本的には並置である．PascalRecognizerではゴール/プラン/テンプレートの階層は問題独立と考えている．そして問題独立なゴールの構造的な組合せにより問題の解を構成する．この考えに基づきテキスト[7]中の種々の例題の記述を行い，認識実験を行った．その結果プログラミングテキストに出ている程度のトイプログラムであれば，このモデルで処理できることがわかった．但しサイズも大きく，また良形であると考えられる実プログラムに対しては，目的の違いや処理時間の点から考えてこのようなモデルを直接適用することは無理であるし，適当でもない．

PascalRecognizerではプログラム認識を具体的には以下のようにとらえる．

（1）　問題記述に基づきアルゴリズム設計を行い，抽象アルゴリズムを記述する．次にプログラム知識を利用して抽象アルゴリズムを具象アルゴリズムに変換する．最後にプログラミング言語知識を利用してプログラムを実現する．認知科学的な実験[11]によれば，初心者のプログラミングプロセスは必ずしもこうではないが，PascalRecognizerではできたものを認識するという立場からこのモデルを利用する．

（2）　抽象アルゴリズムの構成要素をゴールと，具象アルゴリズムの構成要素をプランと呼ぶ．またプランのプログラミング言語による実現をテンプレートと呼ぶ．どのようなレベルの表現をゴールとするかは最終的には抽象アルゴリズムの記述者の判断であるが，PascalRecognizerではデータの基本操作[6]を一つの基準としている．ALPUSとは異なり抽象アルゴリズムの記述のために後述のようなゴールのいくつかの分割法を提供している．

（3）　問題からどのような抽象アルゴリズムを作るか，抽象アルゴリズムの各要素をどのようなプランで実現するか，更にはプランをどのようなテンプレートで実現するかについては種々の変種が存在する．PascalRecognizerではこれらを問題解法木と呼ぶ一種のand-or木で表現する．

```
{assign the middle value of the array to the key(x) of quicksort}
{modify the array elements between l and r so that the elements
 between l and j are smaller than x and the elements between i and
  r are larger than x}
{if j-l>0 then sort these elements}
{if r-i>0 then sort these elements}
```

(a) the first abstract algorithm

```
{assign the middle value of the array to the key(x) of quicksort}
{assign l to i and r to j}
{repeat the following action while i<=j}
    {increase i until a[i] is larger than or equal to x}
    {decrease j until a[j] is smaller than or equal to x}
    {if i<=j then
        {exchange a[i] and a[j]}
        {proceed one step with i and j}}
{if j-l>0 then sort these elements}
{if r-i>0 then sort these elements}
```

(b) the second abstract algorithm

図3 クイックソートの抽象アルゴリズム
Fig. 3 The abstract algorithm of Quicksort.

(4) 問題解法木とプログラムの照合によるorノードの特定がプログラム認識である.

例えばクイックソートの手続き(a[l]からa[r]を小さい順にソートする)に対する抽象アルゴリズムは図3(a)のようなものである.

更に{a[l]からa[j]はxより小さく,a[i]からa[r]はxより大きいとなるようにa[l]からa[r]の要素を入れ替える}という部分をより詳細なゴールに展開した図3(b)のような抽象アルゴリズムも考えられる.

抽象アルゴリズムにおいて{a[i]とa[j]を交換する}と{iとjを一つずつ進める}のように順次的に進めないといけない部分と,{a[l]からa[j]に2個以上の要素があればそれをクイックソートする}と{a[i]からa[r]に2個以上の要素があればそれをクイックソートする}のように順不同のものとがある.また{i<=jであれば...}のように中に更にいくつかのゴールの組合せの入るものもある.問題解法木ではこのような条件の表現法を提供している.

ゴール{a[i]とa[j]を交換する}に対する実現法であるプランとしては{一つの一時変数を利用して交換する}と{二つの一時変数を利用して交換する}という二つがある.更にプラン{一つの一時変数を利用して交換する}に対するPascalによるテンプレートを考えると,

```
x:=a[i];          x:=a[j];
a[i]:=a[j];       a[j]:=a[i];
a[j]:=x           a[i]:=x
```

の2通りがある.

このようにPascalRecognizerによる問題の記述はゴールをベースとした抽象アルゴリズムの記述と,ゴールの実現法としてのプランおよびテンプレートの記述の二つからなる.そして前者を問題固有の知識ベース(Pベース)と,後者を問題独立の知識ベース(Gベース)と呼ぶ.

3.2 変種の取扱い

どのような問題であろうとも,そのプログラムには多くの変種が存在する.PascalRecognizerでは誤りを含んだプログラムを主たる対象とするため,実現法の上からの変種以外に種々の誤りという面からの変種も考慮の対象としなければならない.実現法の変種には次のような階層が存在する

(1) 語いのレベル

識別子の使い分けの変種である.PascalRecognizerでは単一化操作により変数の同定を行う.

(2) 式のレベル

式(算術式,論理式等)の表現の変種である.この中には単に$a+b$と$b+a$といった交換律のレベルの問題から$a*b+(b-a)*a$と$-a*a$といった簡単化

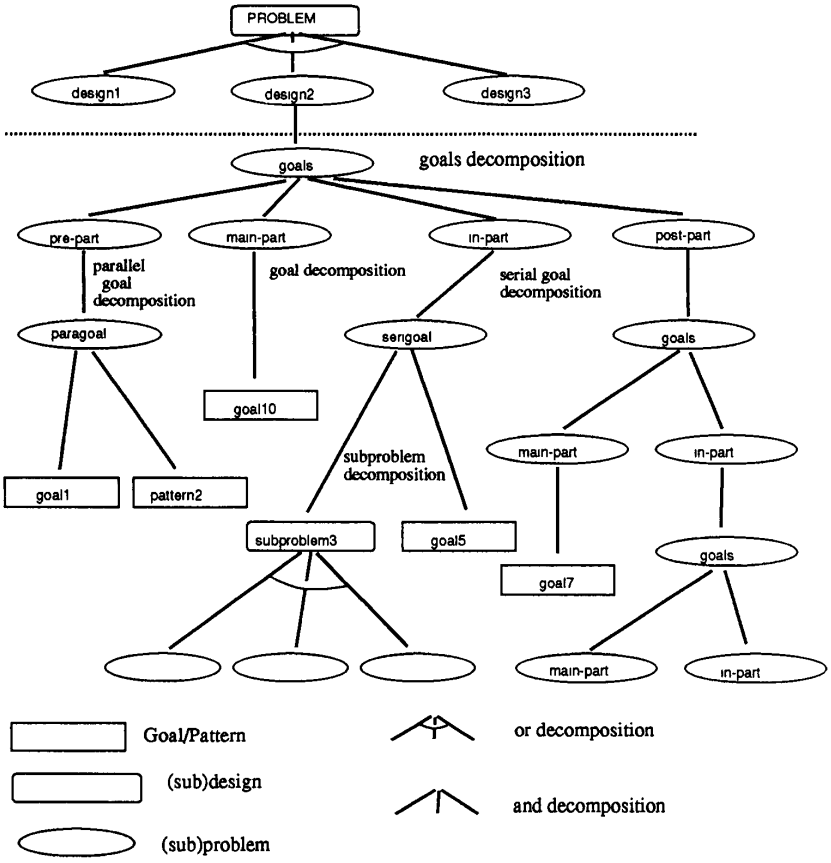

図4　問題解法木の例
Fig. 4　An example of problem solution tree.

を考えなければならないものもある．前者については認識レベルで算術式の式としての等価性判定規則を用意する．後者については解析レベルで標準化，簡単化を行う．

（3）　文のレベル

タスクの文への分解の仕方などである．PascalRecognizerではプランに対するテンプレートの変種として解法木の中に組み込んでおく．また初心者は冗長な文を書くことが多いので，解析レベルでフロー解析に基づく簡単化を行う．

（4）　制御構造／プログラム構造のレベル

どのような制御構造を使うか，また手続き分解を行うかなどであり，PascalRecognizerでは種々のレベルの変種として問題解法木の中に前もって組み込んでおく．

（5）　アルゴリズムのレベル

保守作業へのサポートという面でのプログラム認識では重要であるが，バグの認識という観点からは，問題を課題とする時点でアルゴリズムを指定することも可能であるので，PascalRecognizerではシステムがアルゴリズムを推論するというアプローチはとらず，あらかじめ与えておいたアルゴリズムの候補（設計の候補）の中から，プログラムが利用しているアルゴリズムを同定するという立場をとる．

3.3　Pベース

Pベースは問題固有の記述であり，問題毎に一つの問題の記述といくつかの抽象アルゴリズムの記述（設計記述と呼ぶ）からなる．問題解法木の例を図4に示す．

3.3.1　問題の記述

問題の記述は問題名，問題の説明，問題固有のパターン変数の宣言，正しい設計のリスト，そして誤った設計のリストの五つの部分からなる．

上述のクィックソートに対する問題の記述は以下のようである．

```
(problem-frame
 'problem22
 "クィック法による分類"
 '((variable "?data" "data"
```

```
; まず抽象アルゴリズムとして手続き分割を利用していることを宣言する。
(design-frame
 'design22-1
 "クィック法による分類"
 'problem22
 '(nil nil)
 '((variable "?sort" "sort" "クィックソートを行なう手続き名"))
 '((type procedure-decomposition)
   (procedure-decomposition (main (qsort)))))
;
; 各手続きに対する抽象アルゴリズムの記述
; 主手続き（プログラム本体）の抽象アルゴリズム
(procedure-frame
 'main
 "再帰手続きを利用したクィックソート"
 'design22-1
 '((variable "?i") (variable "?n?") (variable "?a")
   (statement "?statement1")
   (statement "?statement2"))
 ((type goals)
  (goals (main-part (type goal)
                    (goal (name goal65)
                     (apara "?sort" "1" "?n")))
         (pre-part (type goals)
                   (goals
                    (main-part (type goal)
                               (goal (name goal13)
                                (apara "?i" "1" "?n" "?statement1")))
                    (in-part ((object "?statement1")
                              (type goal)
                              (goal (name goal4)
                               (apara "?a[?i]" "rand mod 1000"))))))
         (post-part (type goals)
                    (goals
                     (main-part (type goal)
                                (goal (name goal13)
                                 (apara "?i" "1" "?n" "?statement2")))
                     (in-part ((object "?statement2")
                               (type goal)
                               (goal (name goal19)
                                (apara "?a[?i]")))))))))
```

図5　クィックソートのPベースの記述例

Fig. 5　The Pbase description of the QuickSort program.

```
                "データを記憶した配列")
   (variable "?n" "n" "データの個数"))
 '(design22-1 design22-2)
 nil)
```

この問題は配列?*data*に格納された?*n*個のデータをクィック法で分類するというものである．この場合抽象アルゴリズムとして2種類あることを示している．?*data*等先頭に? の付いた変数はプログラムで実際に使われている文字列と対応づけられる．これをパターン変数と呼ぶ．

3.3.2　設計の記述

ゴールを基本命令として段階的詳細化によるゴール分割により抽象アルゴリズムを記述する．例えばクィックソートの場合ではまず図3(a)のように書く．次にこの中でゴールでないものについて，ゴールを使って更に分割していくか，新たなゴールとして登録するかを選択する．このようなプロセスをすべてがゴールになるまで繰り返す．これをゴール分割と呼んでいる．このように問題の記述をPROUSTとは異なりゴールを基本命令とする抽象アルゴリズムとして与えることにより予測的な認識を可能としている．

設計の記述は設計名，設計の説明，設計固有のパ

ターン変数，そしてゴール分割の記述の四つの部分からなる．アルゴリズムをゴールの組合せとして記述するために，ゴール分割では以下の7種類のゴールの分割法を用意している．

（1） 階層的な分割(goals)

pre-part, main-part, in-part, そしてpost-partに分割する．この形式はゴールの中に成分として更にゴールを含む形態の場合に利用する．例えばある条件を満たすまである処理を繰り返すといったゴールの場合である．次のような記述となる．

```
(type goals)
(goals
   (main-part
      (データを入力し，ある条件を満たすまで
       ある処理を繰り返す))
   (in-part
      (条件に対する記述)(処理に関する記述))
   (pre-part
      (前処理に関する記述))
   (post-part
      (後処理に関する記述)))
```

（2） 順次的な分割(serigoal)

二つ以上のゴールが記述された順序で存在することを要求する．例えばクィックソートでのデータの交換とデータのある添字の値の増減を順次的に行う場合である．

```
(type serigoal)
(serigoal
   (i番目のデータとj番目のデータを
    交換する)
   (iを一つ増やし，jを一つ減らす))
```

（3） 並列的な分割(paragoal)

二つ以上のゴールが順不同で存在することを許す．例えば次のようなデータの和とそのカウントを行うような場合である．

```
(type paragoal)
(paragoal
   (データの和を求めるゴール)
   (データの数をカウントするゴール))
```

（4） サブ問題への分割(subproblem)

既に問題として登録されている問題をサブ問題として利用するような場合である．次のように書く．

```
(type subproblem)
(subproblem （サブ問題))
```

```
(goal-frame
   goal name
   the syntactic category of the goal
   comment
   the description of the goal pattern variables
   the list of legal plan
   the list of illegal plan}

(plan-frame
   plan name
   comment
   goal name
   the description of the plan pattern variables
   the list of legal templates
   the list of illegal templates)
```

図6 ゴールとプランの記述形式
Fig. 6 The description form of the Gbase.

（5） 手続き分割(procedure)

分割の形を固定した手続き分割については認識できる．手続きの本体の抽象アルゴリズムは別のデータとして記述する．

```
(type procedure-decomposition)
(procedure-decomposition
   (手続きの宣言の階層の記述))
```

（6） ゴールでの実現(goal)

例えばデータを交換するゴールの場合，次のように記述する．

```
(type goal)
(goal
   (name 二つのデータを交換する)
   (apara 第1の変数 第2の変数))
```

（7） 直接パターンでの実現(pattern)

算術式等で汎用性のないものは直接プログラムパターンとして記述する．

```
(type pattern)
(pattern
   (pattern パターンの構文的範ちゅう
            パターンの本体))
```

図5にクィックソートの手続き部に対する抽象アルゴリズムの記述例を示す．

図5の記述において例えば

```
(in-part
  ((object "?statement1")
```

```
for each legal abstract algorithm p do
   make error free recognition for p
   if succeeded then return p
endfor
for each legal/illegal abstract algorithm p do
   make program recognition for p
   store the recognition value
endfor
return abstract algorithm whose recognition value is the largest
```

(a)

```
G={main goal}
while G is not empty do
   pick up an element g from G
   select the program fragment p for g
   match p with g
   if succeeded then
      Q = goals for which matching becomes possible
      for each element in Q, bind the corressponding program fragment
      G = G + Q
   else
      look for the goal which has unchecked template or plan
      if such goal exists then
         undo the recognition so far and restart recognition from g
      else
         if g is a primary goal then
            the recognition failed
         else
            set g as an unrecognized goal and proceed
         endif
   endif
endwhile
the recognition succeeded
```

(b)

図7 認識アルゴリズム
Fig. 7 Program recognition algorithm.

```
(type goal)
(goal
 (name goal4)
 (apara "?a[?i]" "?rand mod 1000"))))
```

の部分はmain-partで指定されたゴールgoal13の認識で確定し，パターン変数?*statement*1にバインドされたプログラム断片とgoal4が照合することを述べている．具体的にはgoal13はfor型の繰返しであるが，その本体がgoal4と照合すると述べている．

3.4 Gベース

Gベースはゴールおよび対応するプランを示したゴール部と，プランおよび対応するテンプレートを示したプラン部の二つからなる．図6に示した形式をとる．

4. 認識

認識での処理としては大きく分けて照合，標準化・簡単化・誤り変換，そしての全体の制御の三つからなる．

正しいプログラムとしての認識はfirst-matchで行い，エラーのあるプログラムとしての認識の場合には認識値に基づくbest-matchで行う．プログラム認識の全体のアルゴリズムを図7(a)に，抽象アルゴリズムに対する認識アルゴリズムを図7(b)にそれぞれ示す．抽象アルゴリズムが手続き分割されている場合は手続き単位でプログラム断片を抽出し，それと手続き単位の問題記述と対応づける．誤りを考慮しない認識の場合，照合も誤りを考慮しない照合を行う．

問題解法木上での照合すべきゴールの選択の順序は深さ優先で行っている．またゴール分割がgoals分割の場合はmain-part, in-part, pre-part, そしてpost-partの順で，直列／並列分割の場合は記述の順に選択する．

問題解法木上でゴールはprimary goalとsecondary goalに分けられている．Primary goalはその設計の認識に不可欠なゴールリストであり，secondary goalはprimary goalの認識によりその存在を予測することが可能なゴールリストである．そのためprimary goalに

ついて認識できなかったときは全体として認識できないことになるが，secondary goalについては認識できなくとも本来存在すべきプログラムを示唆することが可能となる．これにより一部の構造だけでも実現しておれば，その範囲内だけでも診断することが可能となる．これを部分認識と呼ぶ．初心者のプログラムには途中まで正常であるが，その後認識の難しい（不可解な）誤りを含むものが多数あり，部分認識は有効である．Primary goalは知識ベース中で図5の5行目の位置で与えるが，この例では与えていない．

5. 照　　合

PascalRecognizerのプログラム認識のための基本操作はプログラム断片とテンプレートとの照合である．例えば図8のようなプログラム断片とテンプレートを考える．図8の例では照合に成功し，パターン変数に対して破線で示したような対応づけがなされている．

プログラム照合における問題点は次の2点である．

（1） 対応するプログラム断片をどう選んでくるか?

問題の記述に使われるゴールは相互に空間的な制約条件をもっている．この制約条件はPascalRecognizerでは階層的なゴール分割として記述する．そしてこの階層関係から照合を行うゴールの順序が決まり，それまでの照合でパターン変数にバインドされた，または残ったプログラム断片が次のゴールに対応するプログラム断片となる．例えば図8の例では照合の結果五つのパターン変数にプログラム断片がバインドされるが，問題記述ではこのパターン変数のいくつかに対して，

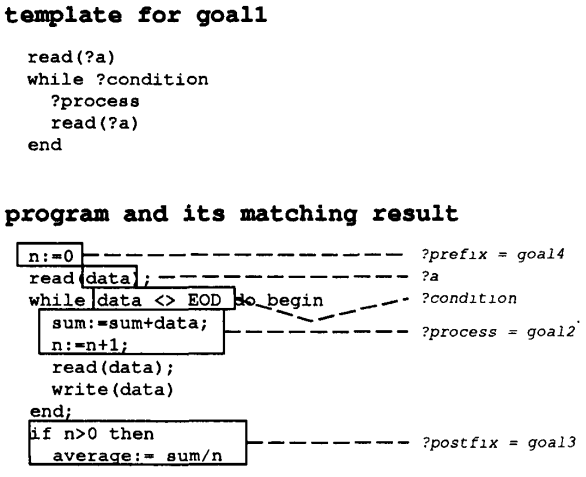

図8　テンプレート照合

Fig. 8　The template matching.

そこにくるべきゴールを指定している．これにより例えばゴール1の前後にゴール4とゴール3がきて，ゴール1の中にゴール2がくることという空間的な制約を表している．この制約条件はPascalRecognizerでは階層的なゴール分割として記述する．そしてこの階層関係から照合を行うゴールの順序が決まり，それまでの照合でパターン変数にバインドされた，または残ったプログラム断片が次のゴールに対応するプログラム断片となる．

（2） 選択されたテンプレートとプログラム断片とをどう照合するか？

プログラム断片とテンプレートがすべて対応づけば問題ないが，良形でないプログラムを扱うPascalRecognizerではそうでない場合をうまく取り扱うことが必要である．これには主に次の二つの場合がある．

（a） テンプレートとプログラム断片との過不足

テンプレートが不足，すなわちプログラム断片が余る場合は一概には誤りとは言えない．しかし最後まで照合できないプログラム断片が残り，それがメッセージのみのwrite文のような冗長なものでなければ誤りである．図8ではwrite(data)が余っている．

プログラムが不足している場合，すなわちテンプレートの一部が照合できない場合は一般的には照合失敗であるが，誤り照合では一部の不足は重みを付けて照合成功とする．

（b） テンプレートの構成要素とプログラム断片の構成要素の間の位置関係の不一致

テンプレートが二つの要素T1,T2の並置からなり，それがプログラム断片P1,P2と照合するとする．そのとき以下のような一部がネストされたプログラムは照合が難しい．

```
P1;                    if .... then
if .... then              P1;
  P2                   P2
```

このような場合については部分照合という照合法により対応する．

6. 診　　断

プログラムの認識結果は問題解法木のorパスを特定したものとしての解法木として表現する．解法木としては以下の情報を含む．

（1） ゴールの認識の有無（認識値）

ゴールと対応するプログラム断片．これにより認識

	the number of recognized programs	the recognition rate
perfect recognition	21	50%
partial recognition	36(including perfect recognition)	85%
recognition failed	6	15%

図9 認識結果
Fig. 9 A recognition result.

```
001   program test1(input, output);
002   const
003       EOD = 99999;
004   var
005       data, number, sum, average: real;
006   begin
007       number := 0;
008       sum := 0;
009       read(data);
010       while data <> EOD do begin
011           if data > 0 then begin
012               sum := data + sum;
013               number := number + 1
014           end;
015       end;
016       average := sum / number
017   end.
      (a) object program for recognition

大域変数の利用状況
"平均値を代入する変数"として"average"を使っている
"番兵"として"EOD"を使っている
"個々の入力データを記憶する変数"として"data"を使っている

"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"number"を使っている
"データの和を代入する変数"として"sum"を使っている

goal"番兵を除いた入力・処理の繰り返し"の実現において次の誤りを犯しています
    "2つのread文の内の後半の繰り返し文中のread文を忘れている"
goal"条件付きの平均"の実現において次の誤りを犯しています
    "numberが0かどうかを検査するガードを忘れている"
```

図10 診断例
Fig. 10 The example of diagnosis.

不可なゴールに対しても本来あるべきゴールを示唆できる.

（2） 認識された場合，対応するプログラム断片，行った変換の種類

（3） テンプレートとプログラムとの間の過不足

（4） 問題および設計のパターン変数へのバインド

現在は「エラーを認識する」という観点から誤りを認識したゴールなり設計なりの単位で診断を行っている．しかしながら「プログラマの意図」という観点だと二つ以上のゴールの実現の誤りがプログラマの一つの誤解から発生している場合もあり，全体としての診断の知識ベースを別にもつことが必要となってくる.

7. 評価，検討

PascalRecognizerはPascalのフルセットをサポートする．ALPUS[6]などは言語知識を独立させ，言語知識を入れ替えることにより他の言語にも適用可としているが，プログラミング教育という観点からは特定の言語に特化して，その指導に専念する方がよいとの

考えから**PascalRecognizer**では言語知識の独立は考慮していない．

プログラム認識の場合，特に手続き分割の変種が問題となるが，抽象アルゴリズムを与えた上での指導という観点から手続き分割は固定と考えている．従って同じものであっても全く異なった手続き分割を行っているプログラムは認識できない．

Pascalを学んで数か月の学生に以下の問題をテストとして与え，提出されたプログラム（42個）を収集し，実現したプロトタイプを評価した．

> **問題**：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，平均には負の数およびEODは含めないものとする．

全体としての認識の割合を図9に，また診断の具体例を図10に示す．実現したプロトタイプは認識が主であり，診断としては間違っているゴールを指摘するだけである．

完全認識の認識率は低いと言えるが，これには前提条件を考慮する必要がある．42個のサンプル中，教師がみても意味不明で診断困難なプログラムが5個あった．また構造的な欠落をもったプログラムもかなりの数に及んでいる．そこで部分認識を導入することにより85％まで認識率を向上させることができた．半数以上は部分的な認識であるが大半は一つのゴールを除いて照合に成功している．対象が言語を学んで数か月（かつテストとして行った）ということで言語概念自体理解できていないこともあり，100%近い認識は無理と考える．

完全認識では対応するゴールが何らかの形ですべて照合されているので，全体にわたっての診断が可能であるが，部分認識ではprimary goalを含む一部のゴールしか認識されないので，診断もその範囲でしかできない．しかしこれは多くの誤りを含むことが原因であるから，逐次的に診断していくという立場からはむしろこの方が（コンパイラ等で言うところの）2次誤りメッセージを避けることができ，初心者に対してはより有効であると考える．また記述および認識という点からテキスト[7]中の30あまりの例題（ヒープソート，クイックソートを含む）の記述および認識実験も行った．但しサンプルおよびバグ知識の問題で多くのプログラムサンプルについての実験は行っていない．

8. むすび

下降型の問題記述に基づき初心者のPascalプログラムの意味誤りを認識し，診断するシステムについて述べた．本システムは下降型のパターン照合に基づく認識および部分照合/部分認識に特徴がある．非常に多彩な初心者の誤りに対して種々の照合法を使うことにより認識を試みた．その結果テストプログラムに対して85%以上の認識率が得られた．認識のできなかったプログラムは人間が見ても意図不明なプログラムであった．

問題解法木という形式での問題（抽象アルゴリズム）の記述の能力について確かめるために文献(7)の例題を中心に約30題の問題を記述し，認識を試みた．この記述に要したゴールの数は約100であった．これによりプログラミングのテキストにあるような例題程度のプログラムに対しては本方式は十分有用であることが確かめられた．

今後の課題としては，以下のものがある．

- 認識の難しいプログラムの多くは複数の誤解(間違い）に基づくと考え得る誤りを含んでいる．そのためそれらを統合して認識することが難しくなっている．プログラミングの指導という観点から考えても，コンパイラの誤りメッセージと同様，逐次的に診断していく方が望ましいと考える．そこでPascalRecognizerでも個々のゴールの認識による診断を取り入れることにより逐次的な診断を実現する．
- 知的チュータリングシステムという立場から学生モデル，教授知識ベース，ユーザインタフェースモジュールを実現する．

文　献

(1) Adam A. and Laurent J.: "LAURA, A System to Debug Student Programs", Aritif. Intell. 15, pp.75-122 (1980)

(2) Johnson W. L. and Soloway E.: "PROUST: Knowledge-Based Program Understanding", IEEE Trans. on Soft. Eng., **SE-11**, 3 pp.267-275 (1984).

(3) Seviora R. E.: "Knowledge-Based Program Debugging Systems", IEEE Software, **4**, 3 pp.20-32 (May 1987).

(4) Harandi M. T. and Ning J. Q.: "PAT: A Knowledge-based Program Analysis Tool", Proc. of Conf. on Software maintenance, pp.312-318(1988).

(5) Murray W. R.: "Automatic Program Debugging for Intelligent Tutoring Systems", Pitman, Readings (1988).

(6) 上野晴樹: "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, SIG-KBS-8902-4(1989).
(7) 土居範久: "PASCAL入門", 培風館 (1989).
(8) Johnson W. L.: "Understanding and Debugging Novice Programs", Artif. Intell. 45, pp.51-97 (1990).
(9) Wills L. M.: "Automated Program Recognition: A Feasibility Demonstration", Artif. Intell. 45, pp.113-171 (1990).
(10) Harandi M. T. and Ning J. Q.: "Knowledge-Based Program Analysis", IEEE Software, **7**, 1, pp.74-89 (Jan. 1990).
(11) 岡本敏雄, 安田恭一郎: "C言語プログラミング過程のメンタルモデルの分析", 日本教育工学雑誌, **16**, 3, pp.119-130 (1992).

(平成6年1月11日受付, 4月5日再受付)

海尻 賢二

昭47阪大・工卒. 昭52信州大・工学部助手. 昭53信州大・工学部助教授. 工博. ソフトウェア開発環境, 知的CAI, 形式的仕様記述に関する研究に従事. 著書「計算機ソフトウェア」(共著, コロナ社) ほか. 情報処理学会, 日本ソフトウェア科学会, 人工知能学会, CAI学会, IEEE, ACM各会員.