コンピュータと教育　10－12
(1990. 4.25)

INTELLITUTORにおける学生モデル構築について

Student Modelling for a Knowledge Based Programming Environment INTELLITUTOR

矢野正己* 村山浩* 酒匂英彰** 石田剛史*** 上野晴樹
Masami YANO, Hiroshi MURAYAMA, Hideaki SAKO, Tuyoshi ISHIDA and Haruki UENO
*東京電機大学　** 現　日本電気（株）　*** 現　日本石油（株）
Tokyo Denki University

あらまし　：　知的プログラミング支援環境INTELLITUTORには知的ＣＡＩ(TUTOR)機能が組み込まれている．学生の状態を理解し，適切な教育的な配慮に基づく支援ができないかというアプローチからである．そこで，我々はプログラミング過程の学生の状態を理解し，学生の犯したエラー，また学生のプログラミングにおける問題解決を教師が学生もイメージしたモデルを構築しようと試みている．学生モデルはプログラミングの過程を表すプログラミング履歴モデル，プログラム理解の結果より得られる誤りモデル，そして学生の知識状態をプログラミング知識に基づいて推定する学生知識状態モデルより構成されている。本編ではTUTORに要求される学生モデルの考え方について，また学生知識状態モデルについて述べる．

キーワード　：　知識工学　知識ベース　ＣＡＩ　プログラミング知識
学生モデル　知的プログラミング支援環境

1　序論

学生に対して適切な指導を行うためには学生の状態を適切に理解しなければならない．学生の状態を理解するための方法論，また表現法を研究しているのが知的ＣＡＩの分野で言われている学生モデルの研究である．これまでに我々は教育向きのプログラミング環境について研究してきたがプログラミングの教育といった面から知的ＣＡＩの機能を付加すべきであろう，またプログラミング環境を知的にするためにはシステムが人間教師のように学生の書いたプログラムを理解し，そしてプログラミング過程を監視することにより，学生の状態を理解する機能が必要になる．

プログラミングに関する知的ＣＡＩのこれまでの研究としては，ＭＥＮＯプロジェクトの１つであるBeverly WoolfらによるMENO-TUTOR[7]がある．これは検出された学生のプログラムのバグに基づいて誤りを訂正するために，ソクラテス問答法によって対話して行くことによって学生に訂正させようというものである．対話という手段を用いているために自然言語処理という難しい問題を含んでいると思われる．またBrown J. ReuserらによるGREATERP[6]がある．これはLISPプログラムを書くにあたり，学生の負担を軽減するために学生にできるだけ多くのガイドをしてやり，学生が間違いを起こしたらすぐに適当な診断メッセージを出し，学生にエラーを訂正させるシステムである．また，大槻らによるBOOKシステム[9]でFORTRANを例にとっている．このシステムでの学生モデル構築に摂動法を提案している．また教授戦略にはStrategy Graphを用いているが，このグラフを作成することは困難である思われる．

これらのシステムにはいずれも，学生の状態を理解しようとする学生モデルの考えが導入されている．本編では知的プログラミング環境INTELLITUTORにおける学生モデル構築について述べるが，それには学生のプログラミング過程を監視し，知識の運用過程を把握することが学生を理解するためには重要であることを主張する．また学生モデル構築に当たってこれまで開発されてきた知的ＣＡＩはシステムと学生が質疑－応答を繰り返すことによって構築されているのに対して，本システムでは質疑－応答という手段を極力避けて構築していこうというところに特徴がある．

本編では試作した学生モデルの一部である学生知識状態モデルについての構築のイメージと，処理結果を示す．

では２章で知的プログラミング支援環境INTELLITUTORの枠組みと知的ＣＡＩモジュール（ＴＵＴＯＲ）の特徴について触れ，３章でINTTELLITUTORにおいて学生の状態を理解するにあたりどのような情報が要求さ

れるのかについて，４章では現在までに試作した学生モデルの一部である学生知識状態モデルについて述べていくことにする．最後に考察し，今後の方針について述べる．

2　INTELLITUTORシステムの概要[2]

INTELLITUTORにおける学生モデルはシステム全てのモジュールからの処理結果より構築されるために，INTELLITUTORシステムの概要について説明しておかなくてはいけない．

INTELLITUTORはGUIDE（ガイデッドエディタ），ALPUS（プログラム理解），TUTOR（知的ＣＡＩ）より構成されている．GUIDEはPASCALの構文グラフを中心に，マニュアルやプログラム例等の付加情報を使うことによってコーディングを支援するシステムである．入力されたプログラムは内部表現に変換された後にALPUSに渡される．しかし，ガイド機能を付加したエディタでは学生の能力を正確に知ることは難しくなる．そこで，知的ＣＡＩ機能を働かせる場合には，一般的なテキストエディタとヘルプ機能を使うことによって作動することになる．ALPUSは，GUIDEの処理結果を受取り，アルゴリズムの知識を表している階層的手続きグラフに基づいてプログラムの論理ミスを発見し，学生の誤りに対して包括的な助言を行う[3]．TUTORはINTELLITUTORに埋め込まれた知的ＣＡＩであり，これはALPUSの処理が終了したときに起動されるものである．そしてTUTORの特徴としては，寡黙はシステムであることである．一般的知的ＣＡＩシステムは学生とシステムの間で質疑応答という手段で学生の知識がどのような状態なのかを推定するのに対して，TUTORシステムはGUIDE，ALPUSからの情報を基に学生モデルがほとんど構築されるところにある．また現在のところ，誤りに対して教えるという立場ではなく，より学生に理解してもらうために，深く考えさせるというアプローチをとっている．さらに，これまで開発されてきたシステムは誤りがあるときにだけ知的ＣＡＩ機能が作動していた．いわばtrap的なものを用意しておき，これに学生が引っかかったときに作動していた．しかし，TUTORはプログラミングということを対象としているために，このようなことはない．というのは，大抵の場合学生は適切な表現でプログラムを書くのではなく，冗長であったり，スタイルがよくなかったりし，人が見てわかりにくいプログラムを書くことが多い．このような場合にも，プログラムにはエラーはないのだが，プログラム書法という点から助言を生成できる可能性を持っている．そして，プログラミング環境と一体化しているという点も特徴である．

3　プログラミング環境における学生モデル

プログラムというのは，１つのプログラミング言語においてもたくさん実現方法はある．しかし，誰が見てもわかるプログラムというのは限られてくる．学生はプログラマとしては初心者であり，プログラミングの経験が少ないためにわかりにくいプログラムを書くことが多い．これは目的を果たすプログラムを書けば良いという観点からは満足いく考えかもしれない．しかし，教育的な観点から考えるとプログラミングの経験をいくら積んだところで，人が見てわかりにくいプログラムを書くようではいけない．また，プログラミングの経験を積みプログラミング能力が向上してくると，教師にアドバイスを求める種類も変わり，アドバイスを示す時期も変わってくる．

つまり，プログラムに構文的なエラー，論理的なエラーがあるときはもちろんのこと，エラーがない場合にも適切な助言を提示し，よりよいプログラムを作成できる技術を身につけてもらう必要があるわけである．では，どのような情報があれば学生の状態を的確に把握できるのかについてINTELLITUTORに於て考えてみる．

3．1　GUIDEにおける学生モデル

GUIDEにおける学生モデルというのは，いいかえると構文的観点からの学生モデルである．GUIDEモジュールでは，スペルミスの修正，学生からの要求に応じてプログラミング言語に関する情報を提示するといったことであるから，GUIDEにおいては学生モデルは必要としない．しかし，後で，TUTORが働くときにプログラミングの過程をシステムは把握しておく必要が出てくる．具体的にはプログラミングをしているときに学生は電子マニュアルのどこを参照したのか，プログラム言語の予約語の適用順序，構文テンプレート内でのプログラミング順序，また構文エラーの情報を表す必要がある．これは学生のプログラミング過程を人間教師が観察するように，システムが学生のプログラミング履歴をモデル化するものである．これをプログラミング履歴モデルと呼ぶ．

3．2　ALPUSにおける学生モデル

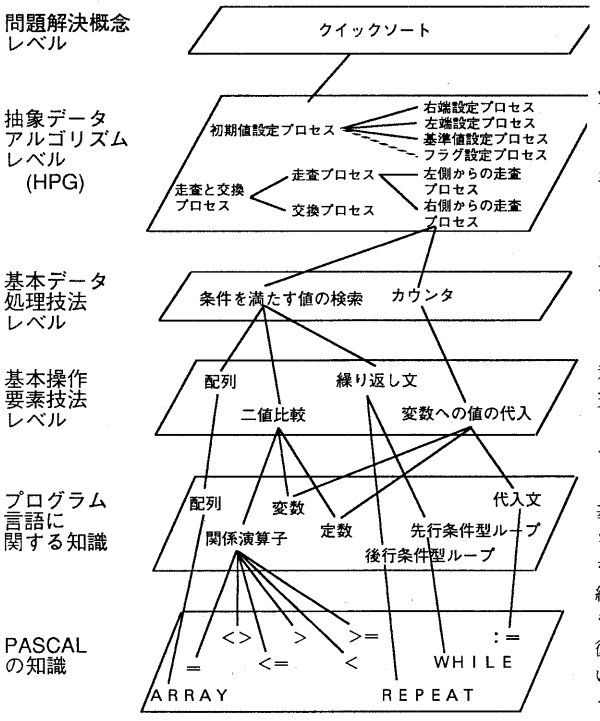
ALPUSモジュールでは，学生の意図理解，誤り原因の同定のために認知実験を通して得られたエラーパターンを階層的手続きグラフに付加情報として格納している．これは，バギーモデルと同類の学生モデルとしてみることができる．このグラフ表現と学生のプログラムをマッチングすることによって論理的に合っているプログラムかどうかを推定している．ここで学生が作成したプログラムの各ステートメントがアルゴリズムではどのプロセスに相当したのか，そしてプロセスが欠如しているのかというALPUSの処理結果がTUTORに引き渡される．このALPUSの欠点としてあらかじめモデル化していた誤りに対しては意図理解し，助言を提示できるが，ない場合には何もできないということになる．そして，助言は包括的なメッセージであり，その学生に適切な助言であるかは保証されない．しかし，

どのプロセスで誤ったのかというレベルまでは少なくても絞れている．ALPUSからTUTORに渡される情報，これを誤りモデルと呼ぶことにする．

3．3　TUTORにおける学生モデル

プログラミング履歴モデルと誤りモデルを統合的に判断し，学生の状態を的確に理解しようというのがここでいう学生モデルである．どのようなイメージなのかについて説明する前に教師の知識であるプログラミング知識について述べておかなくてはいけない．なぜなら，教材知識と対応されることによって学生の知識状態を推定することを試みているからである．

教材知識と一般的に呼ばれている知識はここでは，プログラミング知識となる．プログラミング知識は大きく分けて，プログラミングの技法に関する知識とプログラミングの言語に関する知識に分けられる．プログラミング技法に関する知識は細かな技法的な知識から抽象的な概念的な知識までがいくつかのレベルに階層化できると思われる．具体的な細かな技法に関する知識が最下位にあり，抽象的な概念的知識が最上位にある．これらを下位から上位に向けて表すと，基本操作要素技法の知識とは，変数への値の代入，二つの値の比較，条件分岐などの基本データ操作や制御に関する知識であり，どんな複雑なアルゴリズムでも結局はこれらの基本的操作の組み合せで実現できる．次のレベルの基本データ処理技法の知識とは，二つの変数間の値の交換法，個数の計数法，配列の中の最大値の求め方などが含まれる．抽象データ処理アルゴリズムの知識とは，配列の積を求める手続き，各種整列化の手続きなどのような代表的アルゴリズムに関する実現手続きを含む知識レベルである．最上位の問題解決概念レベルの知識は，配列の積や整列化のような代表的問題解決概念技法の概念やその抽象的アルゴリズムの情報である．一方，プログラミングに関する知識は二段の階層構造をしていると思われる．つまり，いろいろなプログラミング言語に共通の言語要素である基本的なプログラム文に関する知識が下位にあり，個別のプログラミング言語に関する概念的知識が上位にあると思われる．基本的プログラム文に関する知識とは，var,type,array,integerなどの変数宣言文，代入，算術演算，比較演算，論理演算，インデキシング，型変換などのデータ操作文，if文，while文，repeat文，case文などの制御文に関する知識である[1]．このように我々はプログラミング言語に関する知識を分類した．そして，対象として問題はクイックソート，プログラム言語はpascal言語を選び，図－1のように分類化できるものと仮定した．つまり，上級プログラマや教師は階層化されたプログラミング知識を持ち，認知的経済性をはかっているために容易にプログラムを書くことができるのではないか．そして，学生にとってもこのようなプログラミング知識の体系を持つことによって上級プログラマに近付いていくのではないかということを仮定したのである．図－1を説明すると，問題解決概念レベルではクイックソートに関する特徴，例えばソーティング法の中では一番速いソート法である，再帰的手法を用いているということを表している．抽象データアルゴリズムレベルでは，クイックソートを実現するための手続きを表している．これをＨＰＧ（階層的手続きグラフ）というアルゴリズムの実現手続きを表すグラフ表現で表している．クイックソートを実現するには，初期値設定をするプロセス，走査と交換をするプロセス，再帰プロセスよりなりたっている．更に走査と交換プロセスに焦点を当てると走査プロセスと交換プロセスよりなっており，走査プロセスは左側からの走査プロセス，右側からの走査プロセスよりなっている．ここでは以降右側からの走査プロセスに注目し説明する．右側からの走査プロセスに注目した場合，基本データ処理技法レベルでは右側からの走査プロセスは条件を満たす値の検索，カウンタよりなり，条件を満たす値の検索は基本操作要素技法レベルは値の格納，二値比較，繰返し文よりなる．繰返し文はプログラム言語に共通する知識としては先行条件型ループと後行条件型ループよりなり，具体的にpascal言語においては先行条件型ループはwhile文で構成されるという風にクイックソートのプログラムを書くに当たってのプログラミング知識を体系的に表した．



図－1　プログラミング知識

このように教師の知識であるプログラミング知識を

表したのであるが，この知識構造を用いて学生の状態を理解できないか，また学生の誤りがどの程度のレベルの誤りなのかを推定できないかと検討してみた．

これを例を用いて説明すると，左側からの走査プロセスで演算子を「＋」とすべきところを「－」とプログラムを書いたとする．このとき，このエラーは変数への値の代入に関する知識が誤っているとまず，システムは判断する．つまり，基本操作要素技法レベルでの誤りと推定する．そしてプログラミング履歴モデルを参照したときに，学生がエディタの編集機能を用いて他のステートメントからコピーしたとわかったら，これは単なる編集ミスであり，誤りのレベルは最下位のものとして推論し，助言もエラー箇所を示すだけで良いものとして左側からの走査プロセスの説明などは学生には不必要なものと判断する．また，基準値設定プロセスで配列の名前を書かなければいけないのに欠如していて，走査プロセスでのインデックスの示す配列要素と，基準値の比較の二値比較において，

ａ［ｒｉｇｈｔｉｎｄｅｘ］＞ａ［ｂａｓｅ］

のようにプログラムを書いた場合，基準値設定プロセスと走査プロセスについて独立に説明するよりも，おそらく学生は，基準値設定プロセスの役割を何となく理解していたのだが，はっきりと理解していないために起こったエラーであると判断しなければいけない．というのは，これはクイックソート法では一番ポイントとなる知識であり，これがわかっていなければクイックソートがわかったとはいえない．つまり，誤った知識は下位の知識であるのだが，アルゴリズムについて詳しく説明することを要することになる．これもプログラミング履歴モデルで，このステートメントを何度も書き直していたり，カーソルを動かしているとわかれば，ますますアルゴリズムを把握していないものと判断できる．しかし，躊躇することなしにプログラムを書いていたならば，根本的に誤ってアルゴリズムを理解していたと判断することになる．

このように，GUIDEの後ろで学生のプログラミング過程を監視したプログラミング履歴モデルと，ALPUSでどのステートメントが論理的に誤っているのかという情報を得ることによって，学生が誤った原因，またエラーを起こした場合には学生はここでどのような助言をシステムに求めんとしているのかが，わかるものと考えている．つまり，システムが学生の理解状態を的確に理解できるのではないかと考えている．
学生の理解状態を推定した学生モデルが学生理解状態モデルであり，個人学生モデルでもある．

```
program quicksort(input,output);
var a : array [1..10] of integer;
    i : integer;
procedure sort(leftparameter,rightparameter : integer);
      var leftindex,rightindex,base,work : integer;
      begin
            leftindex := leftparameter;
            rightindex := rightparameter;
            base := (leftparameter + rightparameter) div 2;
            while a[leftindex] > base do
                  leftindex:=leftindex +1;
            while base < a[rightindex] do
                  rightindex := rightindex - 1;
            repeat
                  if leftindex <= rightindex then
                  begin
                        work := a[leftindex];
                        a[leftindex] := a[rightindex];
                        a[rightindex] := work
                  end
            until leftindex > rightindex;
            if leftparameter < rightindex then
                  sort(leftparameter , rightindex);
            if leftindex < rightparameter then
                  sort(leftindex , rightparameter);
      end;
begin
      for i := 1 to 10 do
            read(a[i]);
      sort(1,10);
      for i := 1 to 10 do
            write(a[i])
end.
```

図－２　学生が作成したプログラム例

4　学生知識状態モデル

前章では，INTELLITUTOR全体から見てTUTORで，学生の状態を理解するに当たって必要と考えられる情報について，また，この情報をどのように活用し学生の状態を理解しようとしているのかについて述べた．この章では，現在までに試作されている学生モデルの一部である学生知識状態モデルについて述べる．学生知識状態モデルとは，学生がプログラムを書くに当たって使ったと思われる知識を，教材知識と比較することによって正しい知識を用いたかあるいは，誤って理解しているのか，欠落している知識なのかを推定する学生モデルである．したがって，学生の知識の運用ということはプログラミング履歴モデルとの統合をはかっていないため表しきれていない．

学生知識状態モデルはをALPUSからの情報を基に構築するのであるがこれについて述べる．

４．１　学生知識状態モデルの構築

学生が作成したプログラム例を図－２に示す．このプログラムのエラーと説明すると，基準値設定プロセスを書くべき９行目に於て配列の宣言がない．正しくは

```
BASE := A[(RIGHTPARAMETER + LEFTPARAMETER) DIV 2]
```

である．また走査プロセス（右側，左側からの走査プロセス）が誤り，領域を狭めるプロセス（右側，左側）が欠如している．正しくは走査プロセスでは

```
WHILE A[LEFTINDEX] < BASE DO
    LEFTINDEX:=LEFTINDEX+1;（左側からの走査プロセス）
WHILE A[RIGHTINDEX] > BASE DO
    RIGHTINDEX:=RIGHTINDEX+1（右側からの走査プロセス）
```

である．走査領域を狭めるプロセスは

```
LEFTINDEX:=LEFTINDEX+1;（左側の領域を狭めるプロセス）
```

RIGHTINDEX:=RIGHTINDEX+1（右側の領域を狭めるプロセス）
を付加していなければいけない．

このプログラムに対してALPUSは正しくエラーの箇所を発見するのであるが，エラーの情報を図－３のような形で受け取る．

```
(((9 all) process-based-valiable-set)
 ((10 >) process-left-search)
 ((12 <) process-right-search)
 ((19 nil) process-left-narrow)
 ((19-1 nil) process-right-narrow)))
```

図－３　ALPUSからのエラーの情報

これは学生のプログラム内に存在するエラーの数，エラー箇所，エラーの存在するプロセスを示している．ａｌｌはその行全てとALPUSにおけてマッチングされなければならないのにエラーであることを示し，ｎｉｌはその行にあるべきであるプログラムが欠如していることを示す．この情報を用いて，TUTORモジュールで使用する内部表現に変換する．TUTORでの内部表現を図－４に示す．

```
(9 ASSIGNMENT_STATEMENT
   (?????
    (ERROR :=
           BASED_VALIABLE
           DIV
           +
           LEFT_PARAMETER
           RIGHT_PARAMETER
           2)))
(10 AFTER_JUDGE_LOOP NIL)
(11 BEFOR_JUDGE_LOOP
 (COMPARISON_TWO_VALUE
  (ERROR >) (ARRAY A LEFT_INDEX) BASED_VALIABLE))
(12 ASSIGNMENT_STATEMENT
 (COUNT + LEFT_INDEX 1))
(13 BEFOR_JUDGE_LOOP
 (COMPARISON_TWO_VALUE
  (ERROR >) BASED_VALIABLE (ARRAY A RIGHT_INDEX)))
(14 ASSIGNMENT_STATEMENT
 (COUNT - RIGHT_INDEX 1))
```

図－４　TUTORでの内部表現例（一部）

TUTORでの内部表現とALPUSでの処理結果を用いて，ＨＰＧの各プロセスに格納していき，末端のプロセスから上位のプロセスへと結合格納していく．ALPUSでの処理結果とは，学生のプログラムがどのプロセスとして理解されたのかという情報である．この時に，エラーのあったプロセスにエラーの情報をいれ，更にそのプロセスの全ての上位プロセスに同様の情報を格納していく．この段階ではクイックソート形に変換したプログラム のプログラムの中のどのプロセスにエラーがあるかという情報を格納されたことになる．

ここまでの処理で問題解決概念レベル，抽象データ処理アルゴリズムレベルの知識状態を推定したものとする．そして下位のレベルの知識状態の推測であるが，エラーの存在するプロセスに対して，下位のレベルの知識が正しく使われなかった，あるいはその知識が欠落しているとしてそのプロセスの下位の知識に対してこの情報を表すKNOWLEDGEスロットにデフォルトして「０」をセットする．エラーの存在するプロセス全てに対してこの作業が終了するとそれ以外のプロセスは正しく使われたものとし，下位の知識には「１」をセットする．先に「０」がセットされている場合には，その知識は獲得されていると判断し，再度「１」を設定し直します．このように全てのプロセスに於て「０」あるいは「１」が設定されたときに教材知識に基づく学生知識状態モデルが構築されたことになる．

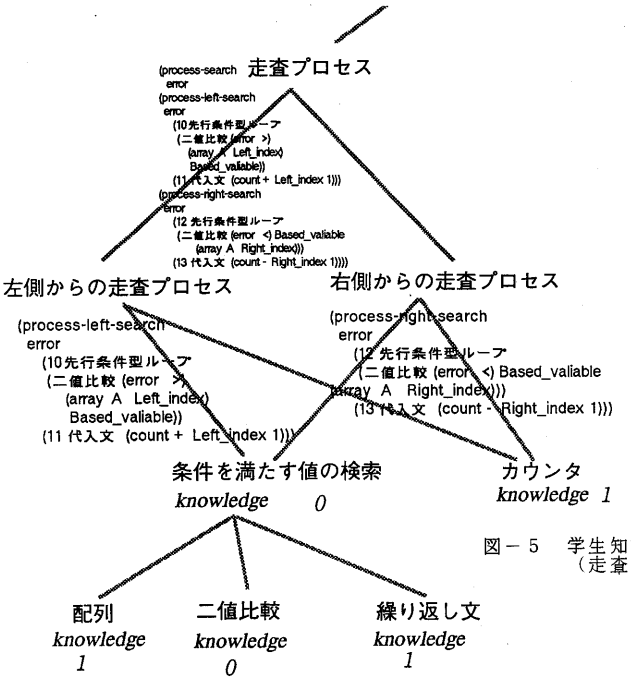
しかし，プロセスが欠如しているために下位のレベルの知識を持っているかどうか判断できないときがある．例えば，クイックソートに於て走査プロセスが欠如していると，基本データ処理技法レベルの条件を満たす値の検出の知識を持っているかどうか判断できない場合がある．この時には，アルゴリズムを表しているテンプレートを示し，それが理解できるかどうか学生に質問することによって確認している．学生知識状態モデルのイメージを図－５に示す．（左側，右側からの走査プロセスでの二値比較のエラーに対して）

5　結論

現在のところ，教材知識としてプログラミング知識の枠組みを整理し，この枠組みに基づいて学生の知識状態を推測する学生知識状態モデルが構築できるように試作した．学生知識状態モデルとは，学生の状態全てを表した学生モデルではなく，学生がプログラムを書くに当たって必要な知識を正しく理解しているかどうかを推定するモデルである．よって，この学生知識状態モデルで知っていないと判断された知識に対しては誤って理解しているものと判断される．

我々がいう学生モデル，つまり学生の理解状態モデルというのは一つ一つの知識が理解されているかどうかを推定するだけではなく，知識がどのようにプログラム作成途中で使われ，どのように問題解決し最終的にプログラムを書いていったのかという過程を把握した上で，学生の状態をシステムが理解したときにそれを学生モデルと呼んでいる．学生の状態理解するためには，プログラミング履歴モデル，誤りモデル，学生知識状態モデルを統合的に判断し，個人学生モデルを構築しなければいけない．

また，学生の書いたプログラムが意味的に正しいかどうかをチェックするプログラム理解システムALPUSにおけるエラーパターンとその誤ったときの原因を認知実験を通して，包括的学生モデルとして充実させることが望まれる．例えば，クイックソートに於て交換プロセスがあるが，ここでエラーを犯した学生に対しては，交換という常識的な知識は理解しているのか，計算機のメモリーのことが理解していないために交換ということをプログラムかできないのか，そしてクイックソートのアルゴリズム上での交換の役割を理解していないのかなど，様々な要因がある．TUTORでは，学生がその学生の誤った原因は何処にあるのかということをプログラミングの過程を考慮した上で推測しな



図－５　学生知識状態モデル
（走査プロセス以降について）

ければならないが，アルゴリズム上での誤り原因はプログラミングの過程例えば注目していた行の時間等を調べることによって，プロセスの注目度が解りプロセス間の理解状態ということも推測できる可能性がある．しかし，常識的な面，計算機の働きなどについて解らないために犯したエラーに対しては質問をせざるを得ないと思われる．その時に典型的なエラーと原因がALPUSのバギーモデルであらわせられるので，質問という手段でもって原因を推定できる．

そして，このTUTORの特徴の一つに教えるという手段ではなく，学生の状態に的をえた助言を提示することによって学生に深く考えてもらおうということがある．それには，学生に同じ助言文を提示したのでは効果はなく，学生もシステムを使う意欲を失うことになる．ここで，助言文を生成する方法を我々は参考文献[3,4]で示したが，この方法では決りきった助言文しか生成できず，堅い文章である．これを柔らかい文でしかも学生理解状態モデルを参照した上で適切な文が生成できるメカニズムを考案する必要がある．

最後に，INTELLITUTOR全体からみたときの知識ベースは，構文的知識はGUIDE，TUTORが，意味論的知識はALPUS，TUTORが参照することになっている．つまり多目的知識ベースでなければならない．このための知識表現を研究していく必要がある．そして，今回は教材知識としてはプログラミング技法に関する知識，プログラミング言語に関する知識に注目して検討したが，プログラミング書法に関する知識，コンピュータに関する常識的知識，また日常生活をするに当たって何気なく使っている知識についても検討を加えていく必要があると思われる．

そして，現在プログラミング履歴モデル，誤りモデル，学生知識状態モデルを統合的に判断した学生理解状態モデルを設計試作中である．

【参考文献】

[1]上野；知的プログラミング環境－プログラム理解を中心に－　情報処理，vol28,no10,pp1280-1296,1987
[2]Ueno.H INTELLITUTOR:A Knowledge Based Intelligent Programming Environment for Novice Programmers,proc COMPCON89 Spring,pp390-395,1989
[3]矢野，上野；知的プログラミング支援環境における知的ＣＡＩ－学生モデルと教材知識についての考察－ AI89-16,1989
[4]矢野，上野；知的プログラミング環境のための学生モデルと説明文生成について，1989信学秋季全大，SA-3-3
[5]矢野，上野；知的プログラミング環境における学生の状態の理解，1990情報処理春季全大，1K-5
[6]Brown J.Reuser,John R.Anderson,Robert G.Farrell:DYNAMIC STUDENT MODELLING IN AN INTELLIGENT TUTOR FOR LISP PROGRAMMING,Proc.of IJCAI 85' pp8-14, 1985
[7]Beverly Woolf,Dvid D.McDonald:Building a Computer Tutor:Design Isuues,IEEE COMPUTER,SEP 1984
[8]中島，上野；アルゴリズムの知識を利用した階層的手続きグラフによるプログラム理解、AI88-31
[9]竹内，大槻；摂動法による学習者モデル形成と教授知識について、情報処理、Vol 28,Vol 1