コンピュータと教育　10－11
(1990. 4.25)

INTELLITUTORにおけるプログラム理解の強化法

A Method of Improvement for Program Understanding Ability in INTELLITUTOR

村山浩* 矢野正己* 関本理佳* 中島歩* 日向野渡** 上野晴樹*
Hiroshi MURAYAMA, Masami YANO, Rika SEKIMOTO,
Ayumi NAKAJIMA, Wataru HIGANO and Haruki UENO
*東京電機大学　　**現　富士通（株）
Tokyo Denki University

あらまし　教育向き知的プログラミング支援環境INTELLITUTORは、人間のチュターのように、知的にユーザーを支援することを目指している。このためには、ユーザーのプログラムを理解する能力が必要となる。プログラム理解の大きな問題点として、プログラムの記述の多様性やエラーの認識があげられる。本論文では、INTELLITUTORにおいてプログラム理解の部分をを担当しているサブシステムALPUSについて、プログラムの記述の多様性とエラーの取扱に重点をおいて述べる。

キーワード　知識ベース　知識工学　知的プログラミング支援環境　プログラム理解
プログラミング知識　階層的手続きグラフ

1．はじめに

現代社会における情報化の急速な進展や、コンピュータの急速な普及にともないソフトウェアの需要が急増してきている。しかしながら、ソフトウェアの開発は依然として人手によって行われているのが現状であり、その生産量や信頼性にはおのずと限界がある。そのために、ソフトウェアの需要に比べて、圧倒的にソフトウェアの供給が不足し、大きな社会問題となっている[1,2]。このような問題に対し、ソフトウェア工学の立場からは、プログラミング言語そのものについての研究や、構造化プログラミング、プロトタイピングなどについてのいくつかの手法が提案され、更に自動プログラミングや、プログラム理解の機能を持ったプログラミング環境などの研究も進められている。

プログラム理解の研究としては、SolowayらがPROUSTでプランと呼ぶプログラムの小さな処理概念を用いて、このプランの組合せによってプログラム理解を行っている[3,4]。またAdamらがLAURAでプログラムの手続きの流れをプログラム・モデルと呼ぶ有向グラフで表現し、プログラム理解に利用している[5]。

我々もこのような背景を踏まえて、知識工学の手法を応用し、既存のプログラミング環境に知的機能を付加し、プログラム理解の機能を持った教育向き知的プログラミング支援環境INTELLITUTORの開発を目指している。本システムは、XEROX1121及びMAC2上に、フレーム型知識工学環境ZEROを用い開発しており、PASCALで書かれたクイックソートのプログラムを対象として研究を進めている。

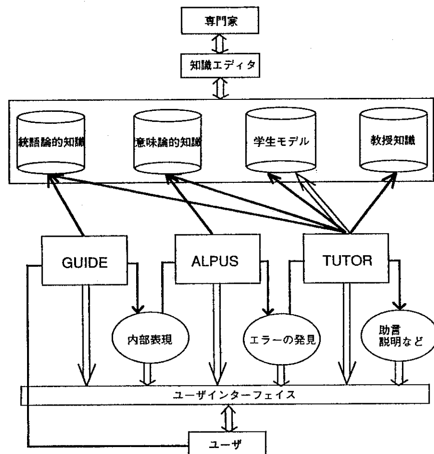
2．INTELLITUTORの概要

我々が開発を進めている教育向き知的プログラミング支援環境INTELLITUTORは、プログラミングに関する各種の知識ベースを持ち、プログラムの作成、修正、及び教育支援を行う対話型の統合化プログラミング環境である。目標としている重要な特徴は次の３つである。

①マニュアルやプログラム例等の各種テキスト情報を構造化ファイルで管理し、エディティン

グ、デバッギング、ティーチング等のモードにおいて、活用できるようにしてあること。
②プログラムの意味理解能力を持ち、ミスを含むコードからユーザの意図を推定したり、複数の論理ミスの指摘と訂正のための助言ができること。
③ユーザの作業を監視し、学生モデルを構築し、これに基づいて教授を行う知的CAI機構を持っていること。

図－1にINTELLITUTORシステムの概要を示す。



図－1　INTELLITUTORのシステム構成

このシステムは、４つの知識ベースと上記の目標を達成するためにそれぞれに対応する３つのソフトウェア・モジュールから構成されている。知識ベースは、構文グラフを中心とした構文則に関する知識を管理する知識ベース、アルゴリズム知識を中心としたプログラムの意味に関する知識を管理する知識ベース、学生の知識と理解状態を表現するための学生に関する知識を管理する知識ベース、及び教授知識を管理する知識ベースとから成っている。ソフトウェア・モジュールは、GUIDE,ALPUS及びTUTORの３モジュールから構成されている。GUIDEはプログラミング用のガイデット・エディタであり、HELP機能を組み合わせたものである。この出力は文法的には正しいものであり、内部表現に変換されてALPUSへ渡される。ALPUSは、アルゴリズムの知識を中心にプログラミング技法とミスに関する知識を用いて論理チェックを行い、論理ミスの指摘、考えられるミスの原因の指摘、及び訂正のための助言を行う。次に、これらの情報からTUTORは学生モデルを構築し、それに基づいて教授を行う。これらを実現するために最も重要な役割を担っている部分が、プログラムの理解機能と学生モデルの構築機能である。本論文では、プログラムの理解機能を担当しているALPUSに焦点を当てて報告する。

３．プログラム理解システムALPUS

３．１プログラム理解の方針

ALPUSにおけるプログラム理解は、アルゴリズムの知識を中心としているが、基本的には認知科学的な考え方に基づいている[6]。ここでいう認知科学的とは、人間のやり方を参考にしてモデル化するという意味である。人間は、与えられたプログラムを読む（理解する）時、そのプログラム・コードの中から何等かの手がかりをつかもうとするはずである。プログラム名、変数名、コメント文等は手がかりを与えるが、プログラム文やプログラムの形（構造）は、より具体的な情報になると考えられる。このようないくつかの情報から、アルゴリズムや実現技法を推論し、また、エラーを含んでいる場合はプログラマの意図を推定し、これらのものをテンプレートとして、プログラム・コードとテンプレートマッチングによって理解しようとするものと考えられる。

我々は、このような観点からアルゴリズムに関する知識を知識ベース化し、モデル推論によってミスを含むプログラムを理解しようと考えている。

３．２ALPUSが扱うプログラムの意味と知識

ここでは、ALPUSで取り扱う理解の対象であるプログラムの意味と知識について述べる。

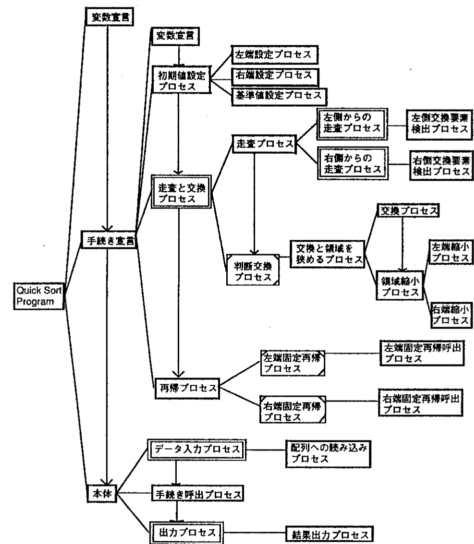
プログラムの意味はプログラミングの知識と対応しており、これらは表裏一体の関係にあるものと思われる。つまり、意味表現と知識表現はほぼ同一の構造をしていると考えられる。一般に、意味はいろいろなレベルで捉えることができるので、まずプログラムの意味のレベルについて考えてみる[7,8]。プログラムにはいくつかの意味レベルが存在するが、これらは抽象度が高いほど問題解決の目的、意図や概念に近づき、低いほど具体的操作に近づく。これらは最

上位から最下位へ、
①問題解決概念レベル
②抽象データ処理アルゴリズムレベル
③データ処理技法レベル
④基本データ操作レベル
の４つに階層化されるものとする。これらはプログラミング知識のレベルとも対応するものと考えられる。但し、プログラミング知識の場合にはプログラミング言語の知識も重要である。なぜなら、プログラミング知識は、
・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
に分類できる。この両者はお互いに独立している。つまりプログラミング技法に関する知識は言語独立である。対応するレベルで考えると、プログラミング技法に関する知識が前述の４つのレベルに対応し、プログラミング言語に関する知識は基本データ操作レベルよりも下位に相当する。従って、プログラム理解では、プログラミング技法に関する知識を利用して、４つのレベルに対応する意味を推定することが基本となる。

一方、プログラム言語に関する知識は、２段の階層構造をしていると考えられる。すなわち、いろいろなプログラム言語に共通の言語要素である基本的プログラム文に関する概念的知識が上位にあり、下位には個別のプログラム言語に関する知識が存在するものと考えれる。ALPUSが対象とする理解は、プログラミング技法に関する４つのレベルのうちの最上位を除く３つのレベルについてである。

３．３階層的手続きグラフ(HPG)

手続き型プログラムにおいては、”プログラム＝データ構造＋アルゴリズム”であることよく知られていることである。よって、プログラムにおいてはデータ構造が対象世界のモデルであるのに対し、アルゴリズムはそのモデルに対する操作にシーケンスであると考えることができる。より具体的に言えば、データ構造を初期状態から目標状態まで更新していく操作のシーケンスがアルゴリズムである。であるから、アルゴリズムは複数のデータ処理のチャンク（まとまり）の組合せになる。このチャンクは、自然に書くプログラムの場合にはシーケンスとなると考えてよいであろう。逆に言えば、シーケンスになるようなところでチャンクに分割することを考える。つまり、アルゴリズムは、プロセスのシーケンスとして表現できる。各プロセスは更に小さなサブ・プロセスの組合せに分解できるので、プロセスの階層グラフとしてアルゴリズムを表現できる。これを階層的手続きグラフ(HPG:Hierarchical Procedure Gragh)と呼ぶ。代表的なクイックソートのプログラムに基づいて書いたHPGの例を図－２に示す。



図－２　クイックソートのHPGの例

各ノードはそれぞれフレームで表現されており、構造や技法などの知識がスロットに格納されている。このようにHPGのような階層グラフを用いることにより、先に述べたレベルの異なる知識をまとめて管理することができる。

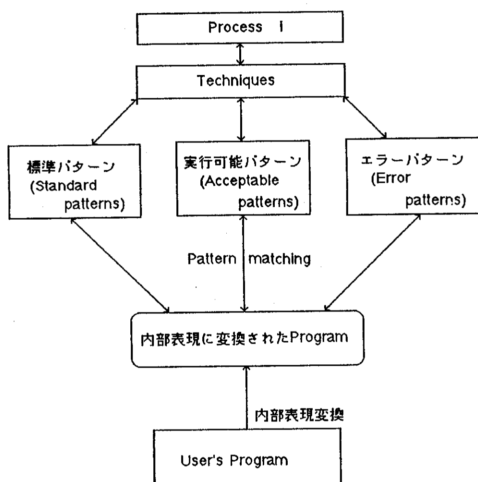
４．プログラムの多様性への対策

４．１プログラムの多様性

一般にプログラムは、ある特定の問題に対して正しいプログラムだけでも数限りなく書くことができる。また、初心者の作ったプログラムにおいては、熟練者には思いもかけないようなバグが含まれている場合があるため、よりプログラム理解を困難にしている。PRUSTでは１つの問題を解くために様々なプランを用意したり、

バグの規則の知識を持つことによってこの問題に対応している。また、LAURAではプログラム・モデルの変換規則を多数用意しており、この規則により学生のプログラムを標準的な形に変換し、対応している。

我々のプログラム理解は、与えられたプログラム・コードを、それと対応するHPGと照合することによって行われる。HPGの各プロセスでは、それを実現する方法がいくつか存在することが普通である。その中で、そのプロセス固有に存在する典型的な方法を、ALPUSではプログラミング技法として管理し、プログラム理解に利用している。ALPUSでは、プログラミング技法は、図－3に示すように、一般的な技法である標準パターン、正しい答えは得られるが教育上好ましくない実行可能パターン、そしてバグを含んでいるエラーパターンの3種類持っている。



図－3　プログラミング技法の管理法

また、プログラミング技法は先に述べたプログラミング技法に関する知識の各レベルごとに管理している。また、プログラミング技法だけでは多種多様なプログラムに対応しきれないため、ALPUSでは、プログラミング技法のそれぞれのレベルに対応した書換え規則を持っている。書き換え規則も、プログラミング技法における実行可能パターンへ変換するものと、バグを含むエラーパターンへ変換するものがある。この書換え規則は、プログラマの作成したプログラムを書き換えるのではなく、プログラミング技法を書き換えるものである。これは、プログラムを書き換えてしまうと、そのプログラムにバグが含まれているとき、誤った書換え規則を適用してしまう可能性があり、適切なメッセージを出せないことがあるためである。以下に、プログラミング技法と書き換え規則について述べる。

4．2プログラミング技法及び書き換え規則

ここでは、プログラミング技法を3．2節で示したプログラミングに関する知識の分類にしたがって、以下に示すように分類する。また、書き換え規則もそれぞれのレベルにおいて示す。

4．2．1抽象データ処理技法

これは、抽象データ処理アルゴリズムレベルに対応するものであり、バリエーションはサブ・プロセスの組合せの差異という形で生じる。例えば、クイックソートのデータ列の分割の方法として、交換法及び挿入法とでも呼ぶべき方法があり、それぞれサブ・プロセスは（初期値設定プロセス－走査と交換プロセス－再帰プロセス），（初期値設定プロセス－走査と挿入プロセス－再帰プロセス）となる。またデータ列の再分割の方法には、再帰を用いる方法と非再帰処理による方法があり、これもサブ・プロセスが異なる。

このレベルの書き換え規則は、書き換えることによってHPGの形が変わってしまうものである。例えば、左・右端固定再帰プロセスで行う、再帰のための条件判定を、手続き宣言プロセスの一番初めに移すものなどがあげられる。

4．2．2データ処理技法

これは、データ処理技法レベルに対応するもので、特定のアルゴリズムによらず、一般的に広く使われるデータ処理の書き方を言う。この技法は、基本データ操作の組合せで表現される。例として2つの変数の値の交換において、標準パターンでは作業変数を1つ用いて3つの代入文で行うものがあり、実行可能パターンでは作業変数を2つ用い4つの代入文を用いるパターンがあり、エラーパターンでは作業変数を用いないで2つの代入文で行うものがある。

このレベルの書き換え規則としては、ループにおいて一般的にはその条件をWHILE文やREPEA

T文の条件式の所に直接書くが、フラグ用の変数を1つ用いて、本来のループの条件をループの中のIF文で判断し、ループの条件式の所ではこのフラグの値を判断するような形に書き換えるものがある。

4．2．3基本データ操作

これは、基本データ操作レベルに対応するもので、プログラム・コードにおとす際の方法である。この例としては、繰り返しにおいてWHILE文やREPEAT文による方法があるが、このようなものは、各プロセス毎に技法として管理するには、記憶効率の点からも好ましくないし、人間もまたそのような記憶の仕方をしているとは考えられない。そこで、ALPUSではこのレベルのものは、各プロセス固有のものでない限り、技法としては持たず、書き換え規則で対応している。

このレベルにおける書き換え規則としては、繰り返しの他に、バグを含むエラーパターンへの変換規則として、代入文の右・左辺を入れ換えてしまうものがある。

4．3プロセスにまたがるエラーの取扱

エラーには1つのプロセス内にとどまるものと、他のプロセスまで考えなければ正確に把握できないものがある。例えばクイックソートにおいて、本来なら基準値設定プロセスにおいてデータ列の中央の値を基準値とし、走査プロセスでこの値を使わなければならない。ところがデータ列の中央の場所を中心にデータ列の分割が行われるとアルゴリズムを誤って覚えてしまった場合、基準値設定プロセスにおいて、データ列の中心を示すポインタを基準値とし、走査プロセスでそのポインタの位置にある値を使おうとする。このような場合、ALPUSではそれぞれのプロセスにおいて、プログラミング技法のエラーパターンを利用することにより発見できる。そしてこのような特定の技法の組合せが生じることにより、異なるエラー原因が考えられるような場合は、そのエラーのあるプロセスをすべてサブ・プロセスとして含んでいるプロセス（この場合は、基準値設定プロセスと走査プロセスを共にサブ・プロセスとして持つ手続き宣言プロセス）でこのような情報を管理し、適切なメッセージを出力するできるようにしてある。

5．プログラム理解の方法

ALPUSにおけるプログラム理解の方法は、与えられたプログラム・コードから得られるいくつかの証拠（主にプログラム構造）に基づいて、抽象データ処理技法の中から適切なものを選択し、インスタンスを生成していく。次に、この結果に基づき、データ処理技法や基本データ操作の知識を用いて、プログラム・コードとのマッチングを取りながらインスタンスを生成していく。各レベルの技法の選択は、まず標準パターンとマッチングを取る。ここでマッチしたらそのプロセスは正しいものと判断する。しかしここでマッチしなかったら、次は実行可能パターンとマッチングを取る。ここでマッチしたら、そのプロセスはあまり好ましくないということが解る。必要とあらば、メッセージを出すことも可能である。ここでマッチしなかったら次には、同じレベルの正しい書換えルールを、標準パターンと実行可能パターンに適用してみる。ここでマッチしたら、実行可能パターンとマッチしたときと同じ扱いにする。ここでもマッチしなかった場合には、さらにエラーパターンとマッチングを取る。ここでマッチしたらエラーをきちんと把握したことになる。マッチしなかったときはエラーの書換えルールを標準パターンからエラーパターンまでマッチングが取れるまで適用する。完全にマッチングが取れインスタンスを生成できたら、完全に理解できたと考える。一部生成に失敗したとき、その部分が理解できなかったものとする。但し、理解できなっかったものでも、いくつかの可能性を提示することによって修正のヒントに利用できるようにしてある。このように、理解力は知識の量と質、及び推論の能力によって決まるが、まだ完全なものではない。

6．まとめ

論理的なエラーの存在するプログラムに対するシステムの診断結果の例を図－4に示す。この例では、システムは3つのエラーを発見している。そこで、システムはそれぞれのエラーに対して、エラーの指摘、訂正のための助言、ミスを犯した原因の指摘を行っている。このように、現在のところ複数のエラーがあっても、ある程度適切なメッセージを出力できるようにな

System-Menu-Window

構文ガイド　段階的詳細化　自由入力　予約語　識別子　Help　マニュアル　終了

Text-Window

```
1   program quicksort ( input , output ) ;
2    var a : array [ 1 .. 100 ] of integer ;
3    i: integer ;
4    procedure sort ( leftparameter , rightparameter : integer ) ;
5     var leftindex , rightindex , base , work , : integer ;
6     begin
7      leftindex := leftparameter ;
8      rightindex := rightparameter ;
9      base := ( leftparameter + rightparameter ) div 2 ;
10      repeat
11       while a [ leftindex ] < base do
12        leftindex := leftindex + 1 ;
13       while base < a [ rightindex ] do
14        rightindex := rightindex + 1 ;
15       if leftindex <= rightindex then
16        begin
17        a [ rightindex ] := a [ leftindex ] ;
18        a [ leftindex ] := a [ rightindex ] ;
19        leftindex := leftindex + 1 ;
20        rightindex := rightindex - 1
21       end
22      until leftindex > rightindex ;
23      if leftparameter < rightindex then
24       sort ( leftparameter , rightindex ) ;
25      if leftindex < rightparameter then
26       sort ( leftindex < rightparameter )
27     end ;
28     begin
29      for i := 1 to 100 do
30       read ( a [ i ] ) ;
31      sort ( 1 , 100 ) ;
32      for i := 1 to 100 do
33       write ( a [ i ] )
34     end .
```

Help-Window

このプログラム内にエラーが発見されました。

===>[基準値設定]プロセスでエラーが存在しています。

エラー内容
あなたのプログラムでは基準値を正しく設定することができません。基準値を正しく設定するためには、次のような代入文を使う必要があります。基準値:=配列[(左端インデックス+右端インデックス) DIV 2]とする必要があります。

あなたは、次のいずれかのミスを起こしたのでしょう。
1:基準値の正しいセットの仕方を知っていたものの、インデックスの指定に気をとられて配列Aを忘れた。

===>[右側からの走査]プロセスでエラーが存在しています。

エラー内容
あなたは、配列A内を13行のWHILE文を使って、配列の右側から基準値BASEよりも大きい値を求めようとしたのでしょう。ところが、配列A内を右側から走査するのにWHILE文内の文では走査できません。右側からの走査プロセスで利用している変数RIGHTINDEXは、この処理の中で1引かれる必要があります。WHILE文内の文には、20行の代入文をそのままコピーして使うべきでしょう。

あなたは、次のいずれかのミスを起こしたのでしょう。
1:12行をコピーし、変数名は変更したものの演算子を変換し忘れた。
2:正しいインデックスの指定の仕方を知っていたものの、12行の代入文につられて演算子-と+を間違えた。
3:無意識に+と書いてしまった。

===>[交換]プロセスでエラーが存在しています。

エラー内容
あなたは、18行と17行を使って値を交換しようとしたのでしょう。
ところが、LEFTINDEXとRIGHTINEXの指し示す値は、は同時に交換はできないのです。
一度、LEFTINDEXの指し示す値を別の値にとっておいて、交換を行いましょう。

あなたは、次のいずれかのミスを起こしたのでしょう。
1:同時に交換できると思った。
2:とりあえずこう書いてしまった。

図－４　ALPUSの実行例

った。しかし、システム側が全く予期しなかったようなプログラムに出会ったときは、それに対して何等かのメッセージも出力することができない。また、現在の推論がプログラムの構造に重点を置いているため、似たような構造を持ったプログラムには適切なメッセージを出力することができないことがある。

今後は、理解能力の向上のために、推論能力の強化と、初心者の作成したプログラムから、エラーに関する知識などを取り入れ、システムの能力の向上を計り、より高度な理解の研究を行っていく予定である。

＜参考文献＞

[1]國友義久：効果的プログラム開発技法、近代科学社（1983）

[2]上野晴樹：知識工学入門、オーム社（1985）

[3]Johnson,W.L. and Soloway,E.:PROUST :Knowledge-Based Program Understanding, IEE Trans.on Soft.Eng.,Vol.SE-11,No.3, pp.11-19

[4]Johnson,W.L.:Intention-Based Diagnosis of Novice Programming Errors,Pitman, London Morgan Kaufman Publishers,Inc., Altos,California(1986)

[5]Adam,A. and Laurent,J.P.:A System to Debug Student Programs, Artificial Intelligence 15,pp.72-122(1980)

[6]上野晴樹、プログラム理解システムALPUSの考え方と方法、人工知能学会研究会資料 SIG-KBS-8903-3

[7]上野晴樹、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、Vol.28,No10,pp.1280-1296,(1987)

[8]Shneiderman,B. and Mayer,R.:Syntactic/ Semantic Interactions in Programmer Behavior:A Model and Experimental Results,Inte.J.of Computer and Inf. Sciences,Vol.8,No.3,pp.219-238(1979)