プランを用いた学習者と教師のプログラムの比較

4U－2

酒井三四郎　陸野慶人
静岡大学工学部

1. はじめに

プログラミング教育の分野において，初心者である学習者の作成したプログラムに含まれる意味的エラーを自動的に発見しようとする試みについて述べる。熟練したプログラマはいろいろな場面ごとに有用となるプログラミング技法を多数身に付けており，プログラム作成に際して積極的に活用しようとする。したがって熟練者の書いたプログラムは一定の形式をしており，ミスが少なく理解しやすい。

これに対して，初心者はそのような技法を知らないので，必要な都度プログラミング言語によって表現を工夫している。したがって表現に一貫性がなく，ミスが生じやすく理解しにくいものになる。

そこで，初心者のプログラムを熟練者の書いた良い構造のプログラムと比較して，誤りがあるか否か，誤りがあればどこにそれが存在し，何がその原因であるかを検出・同定することは重要である。

意味的なエラーを発見するためにはシステムは「正しいプログラム」に関する知識を必要とする。エラーを発見する戦略やその結果はこの知識をどのようにして与えるかに大きく依存している。

LAURA[1)]は正しいプログラムに関する知識をソーステキストの形で持ち，学習者のソーステキストと静的に比較している。多様なプログラム表現を標準的な表現に変換する高度な機構を持っている。しかし，得られた結果は文単位の差でしかないためにエラーの意味付けや原因推定，学習者への助言能力に限界がある。一方，PROUST[2)]はある特定の問題を解決するためのアルゴリズムに関する完全な戦略と，エラーも表現したプログラミング技法（プラン）に関する知識をもとに高度な診断を行った。

我々のアプローチは，教材作成者に負担をかけずにできるだけ多くの問題に対して診断ができるように，問題に依存するプログラム知識として，教師の書いた模範プログラムを用いる。また，多くの教材プログラムを通して学習者に教えようとする普遍的なプログラミング技法に関する知識をプランとして持つ。これらの知識を用いて学習者のエラーを含んだプログラムを診断する。

Comparison between Novice's Baggy Program and Teacher's Model Program Using PLAN
Sanshiro Sakai and Keito Rikuno
Shizuoka University
3-5-1 Johoku, Hamamatsu, Shizuoka 432, Japan

2. プログラム理解のレベル

プログラムの理解を以下の5つのレベルで考える。**(1)基本命令レベル**：プログラミング言語の基本命令でなされている変数の定義，変数への値の代入，2つの値の比較などの操作や条件分岐などの制御を理解するレベル。**(2)データ操作レベル**：複数の基本命令を組み合わせて，ループを用いた個数の計数や2つの変数値の交換のような典型的なプログラミング技法を理解するレベル。**(3)構造レベル**：どのように機能を実現しているかを理解するレベルで，データフローやコントロールフローを理解することに相当。**(4)機能レベル**：さまざまな実現方法を採ったとしても，それらが共通に実現している機能を理解するレベル。**(5)問題解決レベル**：問題領域に関する知識を必要とするが，そのプログラムがはたす役割・目的を理解するレベル。

プランとはプログラムの断片であり，プログラミングにおける典型的な操作のシーケンスを表現するものである。プランはプログラミング知識の一種であり，コーディングにおいてプログラムの意図を実現するための手続きあるいは戦略として利用される。

我々は(3)の知識を教師の模範解答として与え，(1)に対してはプリミティブプラン，(2)に対してはプリミティブプランの組み合わせである複合プランとして与える。(3)～(5)は個々の問題依存の度合いが強く，(1)(2)は問題に依存しない普遍的コーディング・テクニックの知識と考えられる。ただし，(4)(5)については現時点では扱っていない。

3. システムの概観

図1に示すように，教師と学習者のプログラムテキ

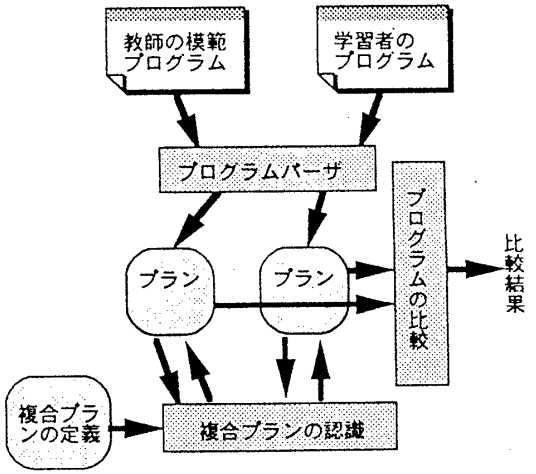

図1システムの概観

ストをプログラムパーザによって見掛け上の記述表現の違いを吸収する。プログラムテキストは基本命令を単位としたプリミティブプランとそれをノードとするグラフに変換される。次に，複合プラン定義に基づいて，上位概念であるデータ操作レベルの概念を抽出する。最後に，それらを比較して診断結果を生成するための情報を引き出す。

4. グラフ化とプランの識別

図2に示すプログラムの断片は図3に示すようなプリミティブプラントそれをノードとするグラフに変換される。グラフは基本的にはノードで操作（基本命令）を，アークで制御の流れを表現する。しかし，これではプログラムの複雑さがそのままフラットにグラフにでてしまうので，まとまったデータ操作を発見することは容易ではない。そこで，図3の破線で示したような包含関係を表すアークを導入している。

複合プランは図4に示すようにプリミティブプランの組み合わせとして定義されている。複合プランの認識はパターンマッチングで行われ，パターン中の変数や定数の同定が行われる。

```
 8    while ((count<10) and (not eoln(value))) do
 9    begin
10        count := count+1;
11        tota l:= total+value;
12        if value > 0 then
13            plus := plus+1;
14        read(value)
15    end;
```

図2 教師の模範プログラムの断片

```
t5:   plan-type: wloop
      statement: 8
      while-cond: (count<10)and(not eoln(value))
      body: (t8 t9 t10 t11)
 :    :
t8:   plan-type: assign
      statement: 10
t9:   plan-type: assign
      statement: 11
t10:  plan-type: if
      t-body: (t12)
      if-cond: value > 0
      statement: 12
t11:  plan-type: read
      statement: 14
 :    :
t12:  plan-type: assign
      statement: 13
```

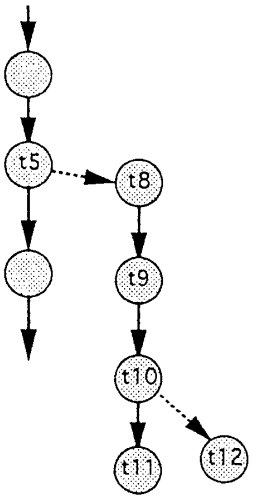

図3 プリミティブプランに変換されたプログラム

◎2変数値の交換

```
c100: plan-type: simple_swap
        assign: ?TEMP := ?VAR1
        assign: ?VAR1 := ?TEMP
        assign: ?VAR2 := ?TEMP
```

◎whileによるカウントループプラン

```
c10:    plan-type: simple_count_wloop
        assign: ?VAR1 := ?NUM1
        wloop:
          begin
            assign: ?VAR1 := ?VAR1+?NUM2
          end
```

図4 複合プランの定義

5. プログラムの比較

プログラムの比較は上で述べた2種類のプランと比較のための経験的知識（戦略）を用いて行う。残念ながら比較戦略は個々のプランのタイプに依存しており十分な形式化が行われていないために，独立した知識とはなっていない。

まず，グラフ上の位置を基本に，大きな単位である複合プラン同士の比較から初める。完全に一致する場合と所定の基準に従って部分的な一致を許す。次に，対応の取れていないプリミティブプラン同士を比較する。ここでも部分的な一致を許す。比較対象の組み合わせはかなりの数に上るが，2つのプログラムは大きくは違っていないとの前提で探索の空間を狭めている。

最終的に，教師のプログラムに存在するプランが学習者のプログラムに存在しない場合，その逆に余計なものが存在する場合，類似のプランが存在するが部分的に一致しない所がある場合を検出する。

6. おわりに

プロトタイプを実現したが，検出した相違点からの助言の生成，データ構造を理解するためのプランの整備，比較戦略の形式化は今後の課題である。また，教師の模範プログラムを解析する際に，システムと教師とで対話を行って，(3)のレベルの正確な理解，(4)のレベルの知識を追加する機構を付け加えることが可能であると考えられる。

本研究は文部省科学研究費補助金（課題番号05780161）の援助のもとに行われた。記して謝意を表する。

参考文献

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

2) Johnson, W. L. and Soloway, E : PROUST: Knowledge-Based Program Understanding, *IEEE Trans. Softw. Eng.*, Vol. SE-11, No. 3, pp. 267-275 (1985).