ノート Note

初学者の作成したプログラムの理解について

川 崎 治 夫*

Understanding Novice Programs

HARUO KAWASAKI*

synnopsis: Program transformation is a basic technique in software engineering, and dependence graphs are useful in compiler optimizations. This paper presents the interactive system which evaluates novice programs. This system makes use of these. The second are used in checking semantic invariance. Combining these can give more powerful system.

要旨：プログラム変換はソフトウェア工学においては基本的な手法であり，dependence graph はコンパイラ・オプチマイゼーションにおいて有効な方法である。今報告においては，これらの手法を使った初学者の作成したプログラムの評価支援システムについて述べる。dependence graph は意味の不変性の検査に使われる。これらの手法を利用すると，より強力なシステムが構築できる。

1. はじめに

プログラミングの授業においては，通常，教師は学生に対してプログラム作成の宿題を出しレポートとして提出させる。現在この過程つまり学生が宿題のプログラムを作成するのを援助し，かつ教師のレポート採点を援助するシステムについて考案中である。今報告においては，まず第一段階としてレポート採点支援システムについて述べる。授業において宿題のプログラムが与えられレポートに対する評価が得られるまでの流れを考えてみると以下のようになる。

Ⅰ．学生は授業で得た知識を基にし，かつシステムの援助を受け宿題のプログラムを作成する。

Ⅱ．学生はプログラムとそれを作るためのプランそして使用した変数の意味をファイルに格納しておく。

Ⅲ．宿題の締切日がきたら教師は宿題の模範解答とそれを作るためのプランそして使用した変数の意味を入力しファイルに登録されている学生のプログラムをシステムの支援を受けながら対話的に処理する。そして宿題に対する評価を与える。

今報告においては主にⅢについて述べることになる。レポート採点支援システムの報告例は

* 国士舘大学電子計算機センター助教授
Associate professor at Computer Center, Kokushikan University

少ない。しかし問題を分析してみるとプログラムの標準形，プログラム理解，ICAI 等分野との関連があることがわかる。

2.　レポート採点支援システム

このシステムは，主に初学者の作成した PASCAL 風プログラムの採点支援を行なうものとする。初期の人工知能の教育への応用として，予め格納された答と学生の答を直すための批評を使って，学生の答に対して応答するという方式があった。しかしプログラムについていえば，1つの答に対して無限個の同値なプログラムが一般には存在するので答をすべて最初に登録することは不可能である。しかしシステムに教師が与えるものになるべく少なくしておきたい。そこでこのシステムでは，教師も学生もプログラムだけでなくプログラム作成のためのプランと使用した変数の意味も一緒に入力する。そこで教師解と学生解を比べる前に

① 両者のプログラムのプランを比べる。

② 両者のプログラムの変数の意味を比べる。

という過程が入る。この段階で相違点が発見された場合は，そのプログラムに特別の印をしておき後程再点検する。内容的には非常に独創的なプログラムである可能性もあれば非常に間違っている可能性もある。今報告は上記の過程においては相違点が発見されなかったプログラムのみを取り扱う。実際のレポートについていえば①②の段階で相違点の発見されないものが割合としては一番多いと思われる。結局教師は多くの場合非常に似通った多くのプログラムを評価することになる。この際評価基準としては，まずプログラムが正しいかどうかという点の判断を優先するものとする。このシステムにおいてはプログラム変換と変数依存解析の手法を取りいれる。

2.1　プログラム変換と変数依存解析

プログラム変換はソフトウェア工学においてよく使われる手法である。このシステムにおいては，教師解と学生解をくらべるときお互いのプログラム変換によって変換し評価のし易い形式にするという方法をとる。

一般にプログラム変換とは，プログラム P から Q への変換で，プログラム P の意味 I_p とプログラム Q の意味 I_q が変わらないものをいう。

プログラムの意味については，プログラム P の入力集合 I から得られる出力集合 O がプログラム Q の同じ入力集合 I から得られる出力集合 R が O と等しいとき I_p と I_q は同値である。

システムはプログラム変換のカタログを持っているものとする。現在のところ変換パターンの選択は全自動的には行なわないものとする。一般にプログラム変換においては変換の方向性

が問題になるが，この場合においては教師解と学生解の距離を縮めるように変換すればよいわけであるから問題とならない。変数依存解析は本来コンパイラ・オプチマイゼーションの為に開発された[3]手法であるが，このシステムにおいては若干変更して文の入換えがなされたときの意味の不変性の検査に使う。文の入れ換えに伴う意味の検査は，2つのプログラムが文を入れ換えただけの違いしか持たない場合，あるいはプログラム交換を繰り返した後等によく出てくる。プログラム変換で足らない部分を変数依存解析によって意味の検査を行なうことになる。また記号実行の機能をもっていてこれを使用できるものとする。

2.2 Dependenece Graph

与えられたプログラムの変数依存解析をするためには，このシステムはプログラムより dependence graph を作成する。graph のノードは while, repeat, for ループの heder または代入文を表わす。graph のアークはプログラム・コンポーネント間の次の4つの関係の内のどれかを表現しているものとする。プログラム・コンポーネントとは代入文または，上のループの header を意味する。

(a) プログラム・コンポーネントをC，ループの header をLとするとき，もしCがLを header とするループの中に存在するかC=Lであるとき，CはLに関して loop dependent という。これをⓁ⋯▸Ⓒと表わす。

(b)

R：X：=⋯ 左の様に定義の連続である場合

S：X：=⋯ Ⓡ—•→Ⓢと表わす。

特にR：X：=⋯がループの外からS：Xに対して初期値を与えているときⒸ—•→Ⓢと表わす。

(c)

R： =X 左の様に参照が定義に先行する場合

S：X：=⋯ Ⓡ⇢Ⓢと表わす。

(d)

R：X：=⋯ 左の様に定義が参照に先行する場合

S： =X Ⓡ⟶Ⓢと表わす。

3. 実　　例

例1．データの総和と個数を計算するプログラム

今，教師解として pr01(図1)，学生解として pr02(図2)，pr03(図3) が登録されているも

```
      begin
A1 :      sum:=0;
A2 :      count:=0;
A3 :    read (new)
W1 :  while not (new=9999) do
        begin
A4 :        count:=count+1;
A5 :        sum:=sum+new;
A6 :        read(new)
        end
    end
```

図1

```
      begin
A1 :      sum:=0;
A2 :      count:=0;
A3 :    read (new)
W1 :  while not (new=9999) do
        begin
A5 :        sum:=sum+new;
A4 :        count:=count+1;
A6 :        read(new)
        end
    end
```

図2

```
      begin
A1 :      sum:=0;
A2 :      count:=0;
A3 :    read (new)
W1 :  while not (new=9999) do
        begin
A6 :        read(new)
A5 :        sum:=sum+new;
A4 :        count:=count+1
        end
    end
```

図3

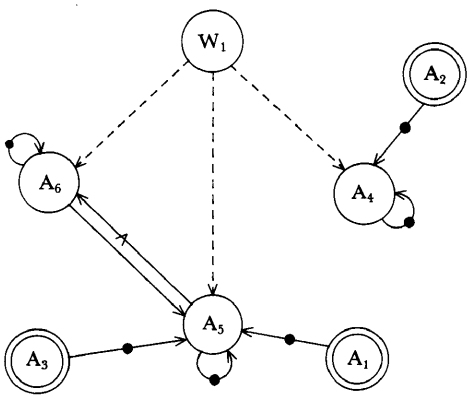

図4

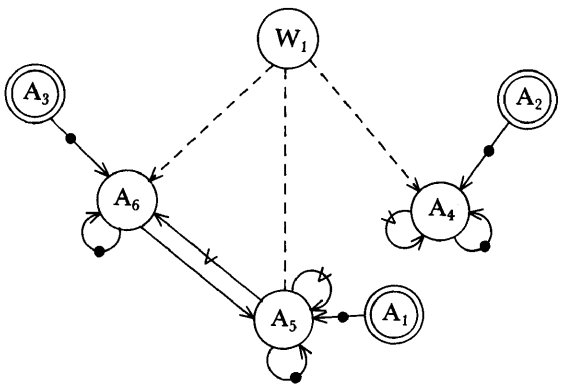

図5

のとする。まずpr01とpr02を比べる。明らかにsum:＝･･･とcount:＝･･･を入れ換えただけである。そこでpr01のdependence graphを作らせる（図4）。A_4とA_5は全く依存関係がないので順序を変えても依存関係に変化はない。よってpr02は正しいと見なされる。次にpr01とpr03を比べてみると文の並べ換えにより構文上は等しくなる。そこでpr03のdependence graphを作らせる（図5）と，A_3とA_6，A_3とA_5の間の依存関係に変化が生し，意味に変化が起きたことがわかる。実際，最初のデータが無視され総和に含まれなくなる。

結局pr02は正解とみなされるがpr03は正解とみなされない。

例2．何日間かの降雨量のデータがある。このとき平均降雨量，最大降雨量，降雨日数を求めるプログラムを作成する。ただし9999というデータが読み込まれたら入力を止める。またこのデータは平均を求める為のデータには含めないものとする[4]。

教師解としてpr03（図6），学生解としてpr04（図7）が登録されているものとする。pr03とpr04の相違点は点線部であることがわかる。そこでpr03についてrainに関する2つのif文を

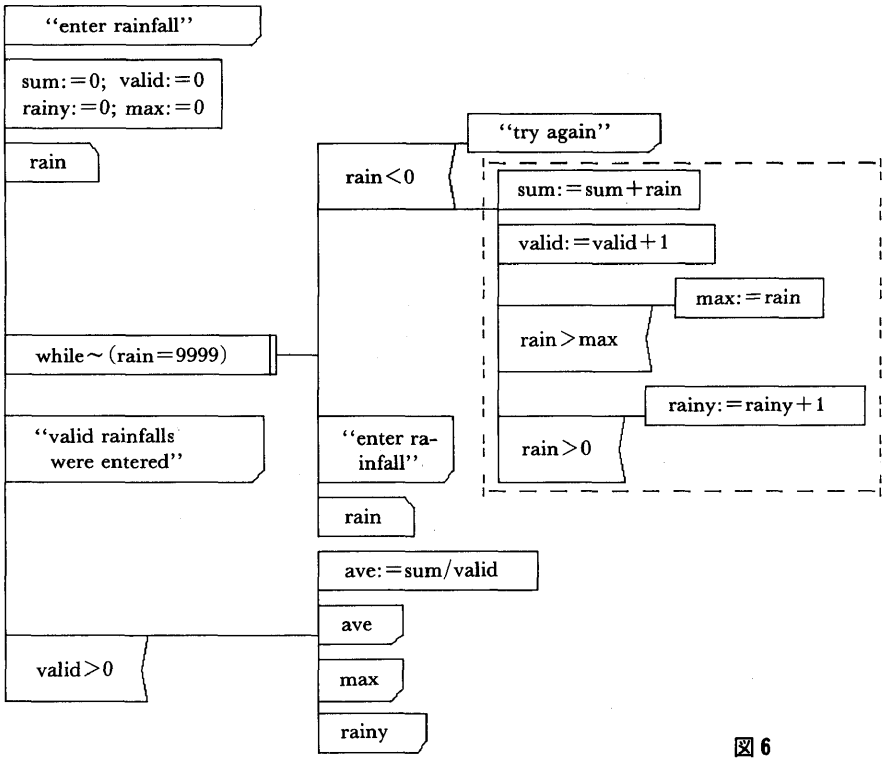
"enter rainfall"
sum:=0; valid:=0
rainy:=0; max:=0
rain
while～(rain=9999)
"valid rainfalls were entered"
valid>0
rain<0
"try again"
sum:=sum+rain
valid:=valid+1
max:=rain
rain>max
rainy:=rainy+1
rain>0
"enter rainfall"
rain
ave:=sum/valid
ave
max
rainy

図 6

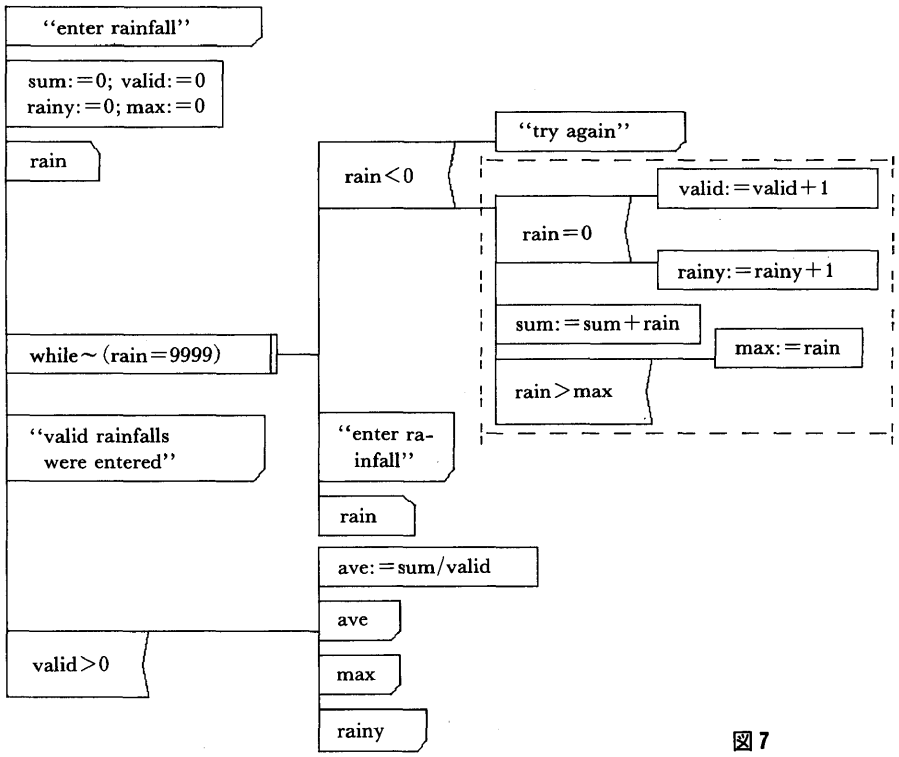
"enter rainfall"
sum:=0; valid:=0
rainy:=0; max:=0
rain
while～(rain=9999)
"valid rainfalls were entered"
valid>0
rain<0
"try again"
valid:=valid+1
rain=0
rainy:=rainy+1
sum:=sum+rain
max:=rain
rain>max
"enter rainfall"
rain
ave:=sum/valid
ave
max
rainy

図 7

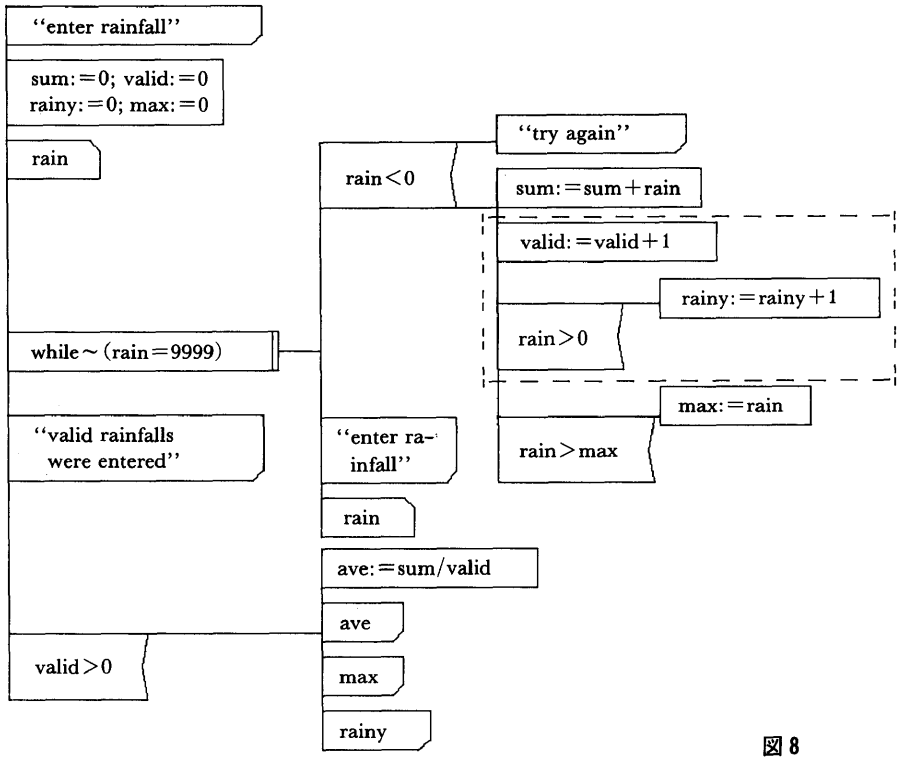
"enter rainfall"
sum:=0; valid:=0
rainy:=0; max:=0
rain
while～(rain=9999)
"valid rainfalls were entered"
valid>0
rain<0
"try again"
sum:=sum+rain
valid:=valid+1
rainy:=rainy+1
rain>0
max:=rain
rain>max
"enter rainfall"
rain
ave:=sum/valid
ave
max
rainy

図 8

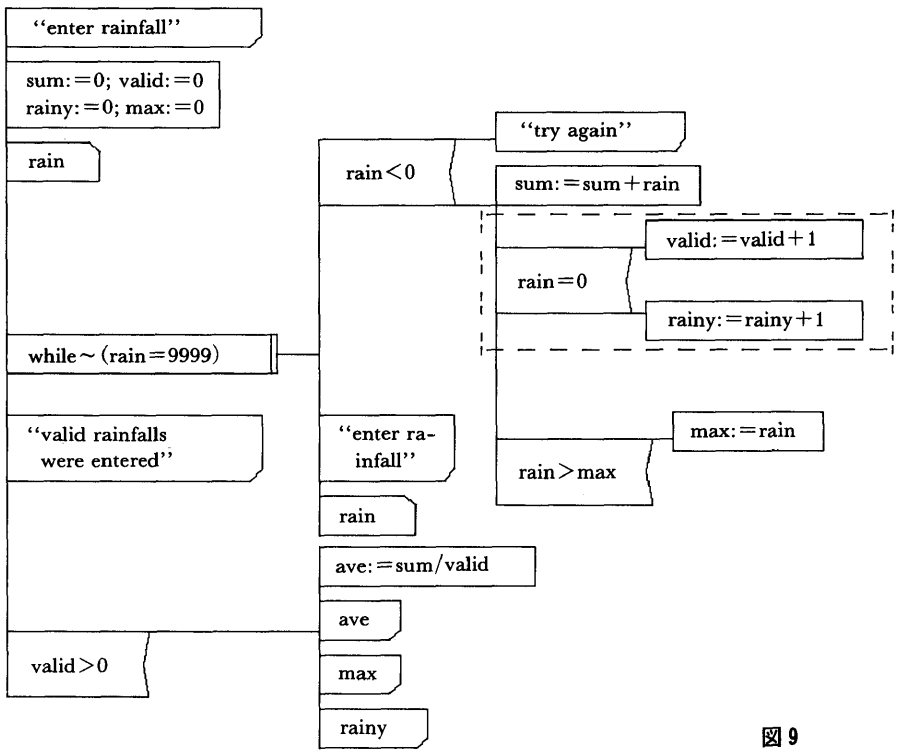
"enter rainfall"
sum:=0; valid:=0
rainy:=0; max:=0
rain
while～(rain=9999)
"valid rainfalls were entered"
valid>0
rain<0
"try again"
sum:=sum+rain
valid:=valid+1
rain=0
rainy:=rainy+1
max:=rain
rain>max
"enter rainfall"
rain
ave:=sum/valid
ave
max
rainy

図 9

入れ換え pr03a（図 8）を得る。この入れ換えは変数依存解析をすれば明らかに rainy:＝･･･ と max:＝･･･ とは互いに関係がないので意味は変化しない。pr04については sum:＝･･･ と直後の if 文を入れ換えるようにして pr04c（図 9）を得る。この入れ換えは変数依存解析により sum:＝･･･, valid:＝･･･, rainy:＝･･･ が互いに関係がないので意味は変化しない。

そこで pr03a と pr04c を比べてみると点線部に相違がある。この部分に関しては記号実行させてみる。すると pr 03a においては rain≧0 のとき valid:＝･･･ を実行し rain＞0 のとき rainy:＝･･･ を実行することがわかる。pr04 c については rain＝0 のとき valid:＝･･･ を実行し，rain＞0 のときには rainy:＝･･･ を実行することがわかる。これから valid の意味が変化していることがわかる。よって pr04は正解ではないことがわかる。

4. ま と め

今報告においてはレポート採点支援システム構想，特に，そこで採用される基本的戦略について述べた。机上でのシュミレーションによってわかったことは以下のとうりである。

① プログラム変換は有効な手段であるが，パターン化しているので実際の場合には必ずしも適応性が高いとはいえないが，変数依存解析を併用すると高くなる。

② 実際のプログラムに対する標準形を定める問題は，見かけとは違い案外難問である。

しかし変数依存解析によって意味が不変であるプログラムは同じものと考えれば標準形に対する 1 つの指標ができたことになる。これについては別の機会に報告する予定である。今後の課題としてはプランの記述方法，プランの比較，プログラム変換パターンの定式化，トータルなシステムとしての構成の在り方，プログラムの標準形の定式化等がある。

参 考 文 献

1) 安村，梅谷，堀越，自動ベクトルコンパイラにおける部分ベクトル化の方法，情報処理，Vol. 24, No. 1, 1984.
2) 安村通晃，スーパーコンピュータとそのコンパイラ，bit, Vol. 17, No. 2, 1974.
3) D. Kuck, R. Kuhn, D. Padua, B. Leasure and M. Wolf: Dependence Graphs and Compiler Optimizations, Proc. of the 8th ACM Symp. on Princ. of Programming Language, 1981.
4) W. E. Johnson and E. Soloway: PROUST: Knowledge-Based Program Understanding, IEEE Trans. On S. E., Vol. SE-10, No. 5.
5) C. Rich: A Formal Representation for Plans in the Programmers Apprentice, Proc. 7th IJCAI, 1981.