プログラミング演習における変数の型及び演算を用いた進捗状況把握方法の提案

2014SE030 石元慎太郎　2014SE061 松原茂輝　2014SE098 高畑宏昭
指導教員：蜂巣吉成

1　はじめに

現在，大学ではプログラミングの教育が行われている．具体的には学生数十名に対して，教員 1 人，数名の TA(Teaching Assistant) による演習形式の授業が行われていることが多い．教員が学生に対し効率のよい指導を行うことは演習を円滑に進める上で重要である．しかし，教員と TA で一人一人の学生のソースコードを読みながら教室を巡回する方法では，限られた演習時間の中で学生全員の問題の進捗状況を把握するのは困難である．学生がどの問題でどのように躓いているかを特定するには作成中のソースコードを読むか，学生から質問を受けなければならない．

これらの解決策として進捗状況把握システムを利用する方法が挙げられる．教員が学生の進捗状況の遅れの原因を特定し，それに伴った適切な指導を行うことが出来れば円滑な授業が実現できる．ここでの進捗状況とは，学生全体が演習の問題における重要な処理が正しく記述出来ているかに着目した正解記述への近づき具合のことである．

文献 [1, 2, 3] ではソースコードの行数，コンパイル状況，インデントといった要素に着目した進捗状況把握手法を提案している．しかし，これらの研究は学生がコンパイルをしたソースコードに対して分析を行うものであり，編集途中やエラーとなるソースコードに対しては進捗状況の把握が行えない．文献 [4] では編集中のソースコードに対しても進捗状況の把握が可能であるが，制御構造と条件式を見ても進捗状況を把握出来ないソースコード 1 のような問題には対処ができない．また条件式を見ると進捗状況把握に時間がかかるという課題が残っている．

ソースコード 1: 例 制御構造が現れない関数

```
1    void swap(int *a,int *b)
2    {
3     int tmp;
4     tmp = *a;
5     *a = *b;
6     *b = tmp;
7    }
```

本研究では蟹江ら [4] の研究で挙げられた課題を解決すべく，別のアプローチから研究を行う．我々は変数の型と演算を見ることによってこの問題を解決することができるのではないかと考えた．解析方法は蟹江ら [4] の研究で用いられていた同値類分割を用いた方法を採用する．教員が学生全体の進捗状況を迅速に把握するためには読むソースコードの数が少なく，記述量が少ないことが必要である．同値類分割は読むソースコードの量を減らすことができる点で有効であると考える．本研究ではプログラミングにおいて重要な，変数の値を変化させるという観点に着目し，ソースコード中の演算と変数の型について同値類分割を行うことで進捗状況を把握する．

本手法を用いた進捗状況把握方法の有効性を確認するために，学生の解答途中のソースコードを用いて実験を行った．

2　関連研究

プログラミング演習において学生の進捗状況を把握するためのシステムはいくつか提案されている．

井垣ら [1] は学生のコーディング過程を分析し，可視化して教員へ提示するシステム C3PV を提案している．学生が入力したソースコードの行数，課題ごとのコーディング時間，単位時間あたりのエディタ操作数，エラー継続時間の 4 種類のメトリクスを計測し，学生全体の進捗を比較する．相対的に遅れている学生に対し教員や TA が指導することによって授業の改善を行っている．

長谷川ら [2] は統合リアルタイム授業支援システム IDISS を提案している．リアルタイムに課題を提示し，提出された学生のソースコードについて，コンパイルやインデントなどを自動で評価し通知することができる．また，学生の提出履歴を収集し分析することもできる．これにより人的リソースの問題を解消でき，リアルタイムで課題を作成し理解度に応じた授業を行うことが可能となる．

伏田ら [3] は学生が誤りに陥る際の傾向を明らかにすることを目的とし，コーディング過程を分析している．分析にあたっては，ファイル保存時やコンパイル時などにソースコードを取得し，そのソースコードを基に定性的及び定量的な分析を行っている．定性的な分析としてはプログラミング熟練者による目視を行い，定量的な分析としてはソースコードのトークンに基づく編集距離に着目し傾向を明らかなものとしている．以上の分析結果から学生の進捗状況を把握し，授業の改善を行っている．

蟹江ら [4] は学生の編集中のソースコードを制御構造と条件式に着目し同値関係を定義することで同値類分割を行い，学生全体の解答の傾向を把握し進捗状況を提示する方法を提案している．

これらの研究と本研究との違いを述べる．過去の研究 [1, 2, 3] では学生の進捗状況について，相対的に遅れている学生や誤ったソースコードを記述している学生の特定をすることはできる．しかし，コーディング過程やどのような誤りをしているかを把握することは困難である．また，コンパイル時やファイル保存時などのソースコードを参考にするので，任意のタイミングでソースコードを見ること

は出来ない．

蟹江ら [4] は編集中のソースコードを見ることによって進捗状況の把握を行う．制御構造を見ることで制御の流れについては判断を行うことができるが，条件式を見た場合に分割数が増えてしまいソースコードを見る量があまり減らないので，教員の判断を遅くする要因となる．また，制御構造と条件式を見ても進捗状況を把握することが出来ない問題には対処が出来ないという課題が残されている．

本研究では学生の編集中のソースコードを変数の型と演算に着目し，同値類分割を行うことによって学生の進捗状況を把握する．

3　進捗状況把握方法

3.1　概略

進捗状況把握システムにはソースコードの取得，ソースコードの分析，教員が閲覧するデータの出力という工程が必要である．

ソースコードの取得には WebIDE を利用する．WebIDE はブラウザ上でソースコードのコンパイル，実行，ファイルの提出を行うことができるシステムであり，任意の時間毎に学生のソースコードの取得を行うことができる．ソースコードの取得は自動的に行われるので，学生への侵襲性はない．

本研究ではデータの分析方法として変数の型と演算に基づいた同値類分割を行う方法を提案する．本研究における同値類分割とは，同値関係にあるソースコードを同値類としてまとめることであり，それぞれの同値類に分類されたソースコードの数を分類数，同値類の数を分割数とする．

同値類分割を行うことで読むソースコードの総数を減らすことができるので，少数のソースコードから全体の進捗状況の把握を行うことができる．それぞれの同値類と分類数を示すことで教員の進捗状況把握を支援する．

同値類分割を行うためにソースコードの抽象化を行う．ソースコードの中の次の要素について抽象化を行ったものを抽象化後のソースコードとする．

- 演算，型
- 制御構造
- 関数呼び出し

3.2　ソースコードの補整処理

今回分析を行うソースコードは記述途中のものが含まれており，抽象化を行う前にソースコードの補完処理として波閉括弧が足りない場合に追加を行う．また，正規化処理として同じ処理だが記述方法が異なるものを統一した表記へ置き換える．「a+=b」を「a=a+b」に置き換える，波括弧が存在しない制御文は波括弧を利用する記述方法で置き換るなどが含まれる．

3.3　抽象化方法

3.3.1　演算の抽象化

演算の抽象化では代入文，及び return 文の抽出を行い，変数名を型名に置き換える．代入は変数の値を書き換えるための処理であり，教員が進捗状況の把握を行う上で重要な要素である．

return 文は関数の返す値として重要である．

3.3.2　制御構造の抽象化

本研究における制御構造とは if 文，else 文，while 文，for 文の 4 つを指す．蟹江ら [4] の研究で制御構造の抽出が学生の進捗状況把握において有効であることが示されている．抽象化方法として制御文のシステムの予約語と波括弧を抽出する．

3.3.3　関数呼び出しの抽象化

複数の関数，または再帰関数の作成を求める問題では関数呼び出しを抽出することで学生の正誤を確認することができるので，進捗状況把握を行うことができる．学生が作成する関数のみを抽出し，実引数は型名へ置き換える．

3.4　同値類分割

ソースコード A，B について，演算，制御構造，関数呼び出しについて抽象化したソースコードが一致した場合ソースコード A，B は同値関係にあるとする．

同値関係に基づいて同値類分割を行う．同値類分割結果として抽象化後のソースコードとそれを記述した人数を示すことで教員の進捗状況把握を支援する．

4　実験

4.1　実験概要

蜂巣研究室 3 年生 8 人の学生に問いてもらった問題について，同値類分割を行い次に示す検証を行う．

実験 1　進捗状況が把握できるかの検証

実験 2　元のソースコードと抽象化後のソースコードの正誤が同じかどうかの検証

今回実験に用いた問題は，次の関数を作成せよというものである．

問 1　解の公式を用いて 2 次方程式の実数解を求める関数

問 2　ユークリッド互除法を用いて最大公約数を求める再帰関数

問 3　整数配列の要素を 1 つずつ後ろへ，最後の要素を先頭に返す関数と要素を入れ替える関数

問 4　整数の累乗を求める関数

実験は問 1 から問 3 と問 4 の 2 回に分けて行った．あらかじめ main 関数は作成済みとして，学生が記述した関数のみを分析の対象とする．

表 1: 問 4 の 2 分時点のデータ

//power()
for{
i=i*i
}
	3

//power()
for{
}
	2

//power()
i=1
for{
i=i*i
}
	1

//power() for{	1

//power()
while{
i=i*i+i
}
	1

表 2: 問 4 の 6 分時点のデータ

//power()
i=1
for{
i=i*i
}
return i
	4

//power()
i=i
for{
i=i*i
}
return i
	1

//power()
i=i
for{
i=i*i
}
	1

//power()
i=i
while{
i=i*i
i=i-i
}
return i
	1

//power()
for{
}
return i
	1

表 3: 問 4 の 12 分時点のデータ

//power()
i=1
for{
i=i*i
}
return i
	5

//power()
i=i
for{
i=i*i
}
return i
	1

//power()
for{
i=i*i
}
	1

//power()
while{
i=i*i
i=i-i
}
return i
	1

4.2 実験 1

進捗状況が把握可能かを判断するために同値類分割結果から次のことを確認する．

実験 1.1 同値類分割結果はまとまるのか

実験 1.2 同値類分割結果から進捗状況の把握は行えるか

4.3 実験 1 結果

問 2，問 3 は解答の制限時間により十分なデータを得ることができなかったので，実験の考察は問 1，問 4 のデータを用いて行う．問 1 については 2 名の学生が無解答であるので 6 名のデータとして扱う．

問 1 は 20 分，問 4 は 12 分データを取得しており，問 1 は 5 分，10 分，20 分段階，問 4 は 2 分，6 分，12 分段階のデータを例として抜き出す．これらの時間は各問題の，解答開始直後，すべての学生が記述を終えた時刻，記述を終えるまでの中間の時刻のデータである．

問 4 の同値類分割結果を表 1，2，3 に，問 1 の結果を図 4，5 に示す．表中の型名 int，double は i，d に置き換えられ，ポインタの場合は p_が型名の前についている．Undef は未定義の変数であることを示す．

問 1 の 10 分段階については 20 分段階とほとんど変化がなかったので表は省略する．表の左が抽象化したソースコード，右の数字が分類数である．

4.3.1 実験 1.1 結果

問 4 の結果について 2 分，6 分，12 分のどの時間でもある程度のまとまりがある．問 1 の結果について 5 分，10 分，20 分いずれの時間についても同値類分割がまとまることはなかった．

問 1 の同値類がまとまらなかった理由として，判別式 D を記述する箇所の違いと分母の記述ミスの 2 点がある．判別式 D の値をあらかじめ求めて変数に代入する学生と，直接解の公式の計算に記述する学生がいた．解の公式の分母である 2*a には丸括弧がなければ結果が変わってしまうが，学生へ提示した実行例の a の値が 1，もしくは-1 であったので，実行例については正常に動作してしまい，間違った記述のまま解答を終える学生がいた．これらによって記述方法が分かれてしまった．

以上より，演算が単純な問題については同値類分割はまとまるが，演算が長くなる問題については記述が複数ありまとまらないことが確認できた．

4.3.2 実験 1.2 結果

問 4 について，開始から 2 分ではまだ記述途中であるが，return 文や初期化式が書けていないことが推測できる．12 分段階では 6 人は正解の記述をしていて，残りの学生は，初期化式がない，return 文がないので間違っていると推測ができる．問 1 について，開始から 5 分ではポインタの記述や解の公式の間違いがあることが分かる．10 分，20 分も同様の間違いが見受けられることから，学生が間違った記述を行っており，どのような理由で正解へたどり着けていないかがわかる．

以上より，進捗状況の把握を行えることを確認できた．

4.4 実験 2

抽象化後のソースコードを確認したときの正誤の判断結果は，抽象化前のソースコードを用いた正誤の判断結果と一致することが望ましい．

正誤の検証を行うために，問 1，問 4 について実験 1 と同様の時間での抽象化後，抽象化前の正誤の判断に差があるか検証する．同じソースコードの抽象化前と抽象化後をそれぞれ目視して正誤判定したものを表にする．

4.5 実験 2 結果

問 1 は 5 分，10 分，20 分段階のソースコード，問 4 は 2 分，6 分，12 分段階のソースコードをそれぞれ正誤判定をして，結果を数値化したものを表 6 に示す．

表 6 より，問 1 と問 4 での正誤の精度は問題ないと考えられる．抽象化前では間違っていたが，抽象化後で正解と判定された問題を確認したところ，for 文内の条件式が間違っていた．目視での誤検出の割合は 5% である．

表 4: 問 1 の 5 分時点のデータ

//QuadFormula()	1
//QuadFormula() *p_d=(-d-sqrt(d*d-4*d*d))/2*d *p_d=(-d+sqrt(d*d-4*d*d))/2*d	1

//QuadFormula()
d=d*d-4*d*d
*p_d=-d+sqrt(d)/(2*d)
*p_d=-d-sqrt(d)/(2*d)
	1

//QuadFormula() *p_d=(-1*d+sqrt(d*d-4*d*d))/2*d *p_d=(-1*d-sqrt(d*d-4*d*d))/2*d	1

//QuadFormula()
d=dˆ2-4Undef
if{
}
	1

//QuadFormula()
d=d*d-4*d*d
p_d=-d+sqrt(d)
p_d=-d-sqrt(d)
return 0
	1

表 5: 問 1 の 20 分時点のデータ

//QuadFormula()	1
//QuadFormula() *p_d=(-d-sqrt(d*d-4*d*d))/2*d *p_d=(-d+sqrt(d*d-4*d*d))/2*d	1

//QuadFormula()
d=d*d-4*d*d
*p_d=(-d+sqrt(d))/(2*d)
*p_d=(-d-sqrt(d))/(2*d)
	1

//QuadFormula()
p_d=(-1*d+sqrt(d*d-4*d*d))/2*d
p_d=(-1*d-sqrt(d*d-4*d*d))/2*d
	1

//QuadFormula()
d=d*d-4*d*d
if{
p_d=0
p_d=0
}
else{
*p_d=(-d+sqrt(d))/2*d
}
*p_d=(-d+sqrt(d))/2*d
	1

//QuadFormula()
d=d*d-4*d*d
*p_d=(-d-sqrt(d))/(2*d)
*p_d=(-d+sqrt(d))/(2*d)
	1

表 6: 正誤の検証結果

抽象化前/抽象化後	問 1	問 4	合計
正/正	4	10	14
正/誤	0	0	0
誤/正	0	2	2
誤/誤	14	12	26

5 考察

実験から演算が複雑な問題ではまとまらないことがあるが，演算が簡単であればある程度まとまって見やすくなる．そして正誤の判定はソースコードを見ることとほぼ変わらないので，全体指導を行うための進捗状況把握方法として問題ないと思われる．

教員の指導例を示す．問 1 の 10 分，20 分段階の同値類分割結果ではポインタの間違い，解の公式の記述を間違えている学生が多い．このことから，教員はポインタについての指導と解の公式の記述内容の見直しの指示を行うことができる．

今回実験を行った問題はいずれも学生の記述が 10 行にも満たないものであり，より長い記述を求める問題についてはまとまらないことが予想される．そのような問題については制御構造のみについて抽象化を行い，おおまかな進捗状況を把握した後に型や演算などの詳細を調べる方法が有効であると考える．抽象化の内容を変更することで問題に合わせた同値類分割を行うことができる．

抽象化後のソースコードの正誤判定方法は現状では教員の目視による判断しかない．正誤判定の支援として，事前に用意された模範解答について，抽象化後が一致したソースコードについては正解であることを示すことによって教員の負担を軽減することができると考えられる．

6 おわりに

本研究ではプログラミング演習における学生の進捗状況の把握を行うために，変数の型と演算に着目しソースコードを同値類分割する方法とそのシステムの設計と実装，及びシステムを用いた実験を行った．

実験の結果，本研究の進捗状況把握方法を用いることで進捗状況の把握を行えることが確認できた．正誤の判定方法の自動化や同値類分割結果の表示方法の実現，及び多くの問題や学習者を対象にした実験による提案内容の評価については今後の課題である．

参考文献

[1] 井垣宏, 齊藤俊, 井上亮文, 中村亮太, 楠本真二 :『プログラミング演習における進捗状況把握のためのコーディング過程可視化システム C3PV の提案』. 情報処理学会論文誌, Vol.54 No.1(2013), pp.330-339.

[2] 長谷川伸, 松田承一, 高野辰之, 宮川治 :『プログラミング入門教員を対象としたリアルタイム授業支援システム』. 情報処理学会論文誌, Vol.52 No.12(2011), pp.3135-3149.

[3] 伏田享平, 玉田春昭, 井垣宏, 藤原賢二, 吉田則裕 :『プラグラミング演習における初学者を対象としたコーディング傾向の分析』. 電子情報通信学会技術研究報告.SS, ソフトウェアサイエンス, Vol.111 No.481(2012), pp.67-72.

[4] 蟹江茉衣香, 松原知奈美, 佐竹玲己衣 :『プログラミング演習における制御構造と条件式を用いた進捗状況把握方法の提案』. 南山大学情報理工学部 2015 年度卒業論文, 2016.

[5] 吉田敦, 蜂巣吉成, 沢田篤史, 張漢明, 野呂昌満 :『属性付き字句系列に基づくソースコード書き換え支援環境』. 情報処理学会論文誌, Vol.53 No.7(2012), pp.1832-1849.