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

2012SE084 蟹江茉衣香　2012SE144 松原知奈美　2012SE219 佐竹玲己衣
指導教員：蜂巣吉成

1　はじめに

現在，情報系の大学ではプログラミングの教育が行われている．教員一人に対して数十名の学生という規模の演習形式が多い中，教員が学生に対して効率の良い適切な指導を行うことは必要である．そのような指導を行うには学生の進捗状況を把握することが必要である．

演習時間内に教員が巡回し，学生すべての進捗状況を把握するのは困難である．学生一人に割ける時間が限られているからである．この解決策として進捗状況把握システムの利用が挙げられ，文献 [1]，[2]，[3] ではソースコード編集履歴やコンパイル・実行結果，エラー履歴などを基に進捗状況の分析を行っている．しかし，これらの方法では学生がどのようにソースコードを記述しているのかというコーディング過程に関する情報の取得・分析を十分に行っておらず，別解や間違った解答の内容まで把握するのが困難である．文献 [4] ではこれらの問題を解決するための研究が行われ，編集途中のソースコードの分析方法を提案している．しかし演算子などの出現回数から大まかな進捗状況は把握できるが，制御構造などは十分に分析しておらず，躓いている原因の特定は難しい．学生がどのようなソースコードを記述しているかを教員が把握できれば，教員は学生に対して適切な指導を行うことができる．また，演習内で使用するので，編集途中のプログラムに対しても一定時間ごとに適切な進捗状況を把握する必要がある．

本研究は，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方法を提案する．進捗状況把握システムとして次の課題がある．本研究の研究対象は前者である．

- ソースコードの分析方法
- 教員への提示方法

進捗状況把握方法として同値類分割による分析を採用する．類似の解答を 1 つの同値類としてまとめることで，学生全体の傾向を把握しやすくなる．本研究では，学生のソースコードの進捗状況を判定するにあたり，基本的な処理の流れを記述できているかという基準に従い，次の観点から同値関係を定義する．

- 制御構造
- 条件式

同一の処理で異なる書き方のものを同値類として扱うために，ソースコードの抽象化を行う．制御構造・条件式を抽象化したものをそれぞれ略式制御構造・略式条件式とする．略式制御構造を用いた同値類分割を行うことで，大まかに学生の進捗状況を把握できる．さらに略式条件式を用いた同値類分割を行うことで，より細かく進捗状況を把握できる．

2　関連研究

プログラミングの演習において，学生の進捗状況を把握するシステムはいくつか提案されている．安留 [1] は，演習課題の採点結果などを学生の席と対応したタブレット端末に表示することで，学生個々の進捗状況を確認可能としている．伏田ら [2] は，トークン化したソースコードと模範解答との差異を進捗状況として図っている．井垣ら [3] は，学生のコーディング過程から 4 つのメトリクスを計測し，学生内で順位づけしたものを表示している．浅井ら [4] は，分析において字句・式・文・文構造の出現回数や組み合わせを計測し，時間経過による数の変動で学生全体の進捗状況を把握している．

これらの研究では学生の進捗状況に対し，他の学生と相対的に見て作業が遅れている学生や，模範解答と比較して悩んでいる学生を把握することができる．しかし，躓いている原因やどこまで記述できているかといった学生の解答の傾向は把握できない．

3　進捗状況把握方法

本研究では，学生のソースコードを同値類分割することで，学生の記述している進捗状況の把握を行う．ソースコード同士の違いを吸収するために，抽象化した制御構造・条件式を用いて同値関係を定義する．編集途中のソースコードも分析対象であるので，構文解析を行うための補正 (4.2 節) をしたソースコードに対して分析する．

ソースコード 1　サンプルコード 1

```
1 int Power(int a,int n){
2         int i, pow=1;
3         if(n>0){
4                 for(i=0;i<n;i++){
5                         pow*=a;
6                 }
7         }
8         return pow;
9 }
```

ソースコード 2　サンプルコード 2

```
1 int Power(int a,int n){
2         int ans=1;
3         int k;
4         if(n>0){
5                 for(k=1;k<n;k++)
6                         ans*=a;
7         }
```

```
8         return ans;
9 }
```

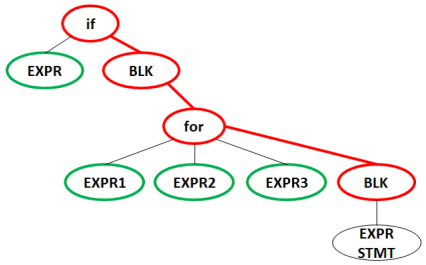

図 1　略式制御構造，略式条件式の図

3.1　制御構造を用いた分析

学生全体の進捗状況の把握には，ソースコードの制御構造の傾向を知ることが有効である．処理の流れを理解しているかで進捗状況に差が出ると考えられるからである．制御構造の中でも制御文や再帰関数は，プログラムの処理の流れを大きく変化させると考えた．これらに関する部分をソースコードから抽出したものを略式制御構造と呼ぶことにする．実際に抽出するのは，制御文の予約語である if，else，for，while と再帰関数名である．

学生が記述した制御構造を調べることで，学生全体の制御構造の記述傾向を把握する．略式制御構造の定義は，抽象構文木から if，else，for，while の頂点と，その頂点間の枝を抽出したものである．ソースコード 1 の略式制御構造は図 3 の赤線の部分となり，ソースコード 1，2 の略式制御構造をテキストで表現するとソースコード 3 のようになる．

ソースコード 3　サンプルコード 1，2 の略式制御構造

```
1 if{for{}}
```

ソースコード 1 と違い，ソースコード 2 は for 文の波括弧が省略されているが，略式制御構造は同じとなる．

略式制御構造を用いた制御構造の同値関係を定義する．

$$pRq \equiv SCS\Big(revise\,(p)\Big) = SCS\Big(revise\,(q)\Big)$$

SCS：略式制御構造 $(SimplifiedControlStructure)$

$revise(p)$：補正後のソースコード p

p, q：ソースコード

p,q は編集途中のソースコードの場合もあるので，revise によって不足した閉じ括弧などが補完される．制御構造のみを用いた分析でも大まかな進捗状況は把握できる．しかし，同一の制御構造でも条件式が異なれば，進捗状況は変化する．

3.2　条件式を用いた分析

制御構造を用いた分析で生じた問題に対し，条件式を用いた分析を行う．条件式を用いた同値類分割は，同値関係に基づく制御構造の同値類の要素に対して行い，進捗状況の差を図るのに必要と判断した要素を抽出したものを同値関係に利用する．この要素である演算子，定数，'\0'，NULL，角括弧 (配列)，丸括弧，カンマ，セミコロン及び再帰関数の引数を抽出したものを略式条件式と定義する．略式条件式は順序付き集合となる．変数名などの基準も考えられるが，変数名が異なれば同値関係に基づく条件式の同値類が不必要に増えてしまい進捗状況の把握が困難になることから採用しない．ソースコード 1 の略式条件式は図 3 の緑線の部分となり，ソースコード 1，2 の略式条件式をテキストで表現するとソースコード 4，5 のようになる．集合の各要素は丸括弧で囲むことで表現している．

ソースコード 4　サンプルコード 1 の略式条件式

```
1 (>0)(=0;<;++)
```

ソースコード 5　サンプルコード 2 の略式条件式

```
1 (>0)(=1;<;++)
```

ソースコード 1 と違い，ソースコード 2 は for 文の条件式の部分で初期値が異なっており，別の同値類とされる．

略式条件式を用いた条件式の同値関係を定義する．

$$pR'q \equiv pRq \wedge \Big(SCE\big(revise\,(p)\big) = SCE\big(revise\,(q)\big)\Big)$$

pRq：略式制御構造を用いた制御構造の同値関係

SCE：略式条件式 $(SimplifiedConditionalExpressions)$

$revise(p)$：補正後のソースコード p

p, q：ソースコード

4　進捗状況分析ツールの設計・実装

4.1　設計

演習での出題形式についての制約を次に示す．どちらも初心者向けの演習としては妥当である．

- 関数名は教員が指定したものとする
- 関数は 1 つとし，main 関数より前に記述されるものとする

一定時間ごとに学生が編集するソースコードをツールを用いて分析し，教員は分析結果から指導が必要かどうかを判断するという使用方法を想定している．一定時間は，演習の場を想定しているので最短 1 分と仮定する．

本ツールは，学生のソースコードの分析にコード断片にも利用できるプログラム解析器 TEBA[5] を用いて，次のような流れで分析を行う．はじめに，ソースコードはコンパイル不可能な編集途中のものが前提であるので，補正を

行う．次に，補正済みソースコードを TEBA で解析し，解析結果を用いて略式制御構造・略式条件式を抽出し，結果を各々 1 つのテキストファイルにリストとして出力する．最後に，抽出されたリストを基に分割を行い，結果を 1 つのテキストファイルに出力する．

4.2 ソースコード補正

TEBA による解析結果が不適切であるので，次のようにソースコードの補正を行う．

- 閉じ波括弧は，main 関数直前に補完する
- 閉じ丸括弧は，不足する行の次の行の先頭に補完する
- セミコロンは，for 文内の不足している数だけ補完する

ソースコードを先頭から順に書く場合が多いことと，実際のデータを確認した結果，開き括弧が不足している学生は稀であったので，開き括弧は補完しない．

5 実験と検証

5.1 実験方法

演習開始後から一定時間ごとに取得したソースコードの同値類分割を行う．学部 3・4 年生 27 名が過去に解いた問題を次に示す．

- ex0001 文字列をコピーする関数
- ex0002 実数の累乗を計算する関数
- ex0003 文字列の長さを返す再帰関数

実験には各問題の開始から一定時間ごとにデータを抽出したものを用いた．制約より，学生が記述する必要のない関数は分析の対象外とする．

5.2 実験 1

ソースコード補正の有効性，及びコンパイル不可能なソースコードに対する分析の妥当性を検証する．実験に用いたデータのうち，コンパイル不可能なものを抽出する．抽出したソースコードにソースコード補正プログラムを適用し，略式制御構造と略式条件式の抽出を行い，抽出の結果が意図通りか確認を行う．また，開き括弧と閉じ括弧の数が等しく，抽出対象物が抽出できている場合，抽出結果を意図通りとする．

表 1 コンパイル不可能なソースコードのツール適用結果

		データ数	ソースコード補正プログラムによる補正	抽出結果
括弧不足	開き波括弧	1		意図通りの抽出
	閉じ波括弧	8	main関数の直前に括弧追加	
	閉じ丸括弧	1	次の行の先頭に括弧追加	
条件式編集途中	if	2		
	while	2		
	for(セミコロン2つ)	3		
	for(セミコロン1つ)	1	セミコロン1つ追加	
	for(セミコロン無)	2	セミコロン2つ追加	
条件式以外のセミコロン不足		12		意図通りの抽出
		1		括弧が不足した抽出
関数の中に関数あり		2		関数内の関数定義を無視した抽出
閉じ波括弧不足+条件式以外のセミコロン不足		2	main関数の直前に括弧追加	意図通りの抽出
閉じ波括弧不足+条件式途中(for, セミコロン1つ)		1	main関数の直前に括弧追加 セミコロン1つ追加	
開き波括弧不足+条件式以外のセミコロン不足		1		
その他 (ポインタの文法ミス・コメントアウトできていない スペルミス・セミコロンなどはあるが編集途中 など)		23		

コンパイル不可能なものは全部で 62 データであった．表 1 より，補正が必要な 15 データは，正しく補正されていた．補正が不必要なデータのほとんどは TEBA の解析結果に問題がなく意図通りの抽出を行えたが，一部のデータが意図通りの抽出を行えなかった．関数内に関数が定義されていたものは正確な構造を把握できなかったが，記述が不適切であるので対応しない．また，一部の抽出対象外の部分でセミコロンが不足しているものは，閉じ括弧が不足した抽出結果となった．しかし，全データ中約 95% は意図通りの補正と抽出を行うことができたので，ソースコード補正は有効，コンパイル不可能なソースコードに対する分析は妥当であると言える．

5.3 実験 2

time コマンドによって 100 名分の実行時間を測定し，想定される時間内に分析可能かを検証する．実行環境は，Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz，6.0GB メモリ，ubuntu 12.04 LTS (32bit) である．

プログラムの呼び出しから終了までにかかった 100 名分の実時間は，ex0001 では 29.355 秒，ex0002 では 30.233 秒であった．どちらとも 1 分未満だったので，想定される実行時間は妥当だと言える．抽出による実行時間が全体の約 90% を占めた．

5.4 実験 3

教員は進捗状況情報を見て学生の進捗状況を判断する．略式制御構造と略式条件式から元のソースコードをある程度推測し，解答の正誤判断を行うことができれば，進捗状況を把握できると言える．実験 3 では略式制御構造と略式条件式の妥当性と，同値類分割による分析方法の妥当性を検証する．実験には ex0002 の全 81 データに対して同値類分割を行い，出力される進捗状況情報を用いる．進捗状況情報から元のソースコードを推測し，推測結果と実際のソースコード比較する．その後，進捗状況情報から解答の正誤判断を行い，実際のデータの正誤と比較する (表 2)．元のソースコードの推測，及び解答の正誤判断は学生 3 名及び教員 1 名で行った．また，正誤判断の適合率と再現率を図 2 の定義に沿って算出する (表 3)．TP=0 または TN=0 の場合，数値は表記していない．

表 2 各時間ごとの正誤判断の結果とその人数

正誤判断		実際の正誤						比較結果	
略式制御構造	略式条件式集合	5分後	人数	10分後	人数	15分後	人数	判断と同じ	判断と異なる
○	○	×	1	○	7	○	9	16	7
				×	5	×	1		
	×	×	4	×	5	○	1	13	1
						×	4		
	△	×	2			○	1		
						×	1		
×		×	20	×	10	×	10	40	0

○：正解であったもの，×：不正解であったもの，△：正誤判断できなかったもの

進捗状況情報から元のソースコードを全て推測することができ，推測の精度は約 85% であった．表 2 より，全データ中約 95% のデータに対して解答の正誤判断が行えた．しかし，表 3 より 5 分後の正解の適合率・再現率と

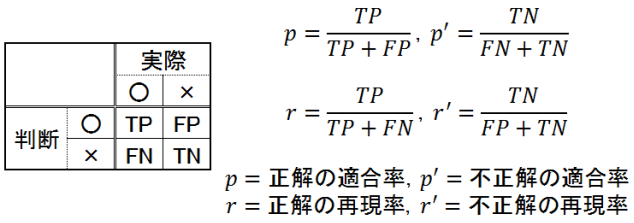

図 2　適合率と再現率の定義

表 3　正誤判断の適合率と再現率

	p	p’	r	r’
5 分後	-	1.00	-	0.96
10 分後	0.58	1.00	1.00	0.75
15 分後	0.90	0.93	0.90	0.93

10 分後の正解の適合率が低い結果になったので，正解と判断したが実際は間違っているソースコードの確認を目視で行った．確認したところ，編集途中または同値関係以外の部分で不正解であるが，制御構造と条件式は正しく記述できている学生が多かった．また，時間経過に伴い正解の適合率・再現率が高くなっているので問題はないと考えられる．

以上より，学生の解答が間違っているかどうかを進捗状況情報から判断することが可能であり，また躓いている原因などのソースコードの内容を大まかに把握できると言える．よって，定義した略式制御構造と略式条件式，及び同値類分割による分析方法は妥当である．しかし，条件式に対しての推測と正誤判断を行うのには時間を要した．この問題に対して，模範解答を用意して一部のソースコードを自動判定するといった対処が必要と考えられる．

5.5　教員への提示方法

2 つの同値関係について分析することで学生の進捗状況を把握できた．しかし記述人数を示すだけでは，一目で進捗状況を把握することは難しい．よって，記述している制御構造と条件式を，略式制御構造と略式条件式として表で示す方法を考察する．本分析の結果から得られる 5 分時点と 10 分時点の制御構造の進捗状況情報を表 4 に示す．5 分から 10 分にかけて正解と判断する制御構造の記述人数が増加していることにより，学生が正解の解答に近づいていることが把握できる．また，制御構造の同値類を選択することで，条件式の進捗状況情報を表に示す．例として，同値類分割の結果 10 分時点でのある条件式の進捗状況情報を表 5 に示す．これにより，条件式の部分で躓いている学生を把握できる．

6　おわりに

本研究では，学生全体の進捗状況の把握を実現するために，ソースコードを制御構造と条件式に着目にして同値類分割する方法を提案した．そのために同値類分割の設定と同値関係の設定を行った．

実験と検証の結果，指導すべき対象や内容の把握，指導の必要性の判断に有効な分析を行うことができ，進捗状況の把握が可能となった．今後の課題としては，今回実験に用いた問題以外のソースコードに対しても検証が必要である．また，模範解答などを用いた推測の負担軽減方法と教員への提示方法実現についても今後の課題である．

表 4　制御構造の進捗状況情報の提示方法

ex0002の5分後の同値類分割の結果

for	if{ for } else{ for }		if{ for } else{ if }	if{ for }
10	4	3	2	2

ex0002の10分後の同値類分割の結果

if{ for } else{ for }	for	if{ } else{ if{ for } else{ for } }	if{ for } else{ if{ } else{ for } }	if{ for } else{ if{ for } else{ } }
9	3	2	2	2

表 5　条件式の進捗状況情報の提示方法

if{ for } else{ for }	(>0) (;>0;−−) (;<0;−−)	1
	(>0) (=0;<;++) (=0;;)	1
	(>0) (=0;<;++) (=0;<;++)	1
	(>0) (=1;<=;++) (=(−)1;>=;−−)	1
	(>=0) (=0;<;++) (=0;>;−−)	4
9	(>=0) (=;>0;−−) (=∗(−)1;>0;−−)	1

参考文献

[1] 安留誠吾，“Web プログラミング演習における学習進捗把握，” 教育システム情報学会 第 39 回全国大会，H3-3，2014.

[2] 伏田享平，玉田春昭，井垣宏，“プロ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，pp．67-72，2012.

[3] 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，pp.330-339, 2013.

[4] 浅井祐太，石川航，松井暁生，“プログラミング演習における進捗状況把握方法の提案，” 南山大学情報理工学部 2013 年度卒業論文，2014.

[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，pp.1832-1849, 2012.