RK-001

関数呼び出しの分析による頻出学習パターンの導出
Derivation of Frequent Learning Pattern Based on Procedure Calls Analysis

谷川 紘平 † 原田 史子 ‡ 島川 博光 ‡
Kohei Tanigawa Fumiko Harada Hiromitsu Shimakawa

1. はじめに

プログラミング初心者である学習者がプログラミング技術を習得するためには，実際にソースコードを記述する演習をしながら学習する必要がある．そのため情報系の大学では，プログラミング演習の科目を設けている．プログラミング演習において，学習者はソースコード作成過程で何度も行き詰まり試行錯誤することで，プログラミングの知識を定着させていく．

ソースコード作成過程で行き詰った学習者は指導者に質問をする．質問を受けた指導者は学習者が理解に苦しんでいる内容を把握したうえで指導する必要がある．しかし，その場で学習者一人一人の理解に苦しむ内容を把握するには多くの時間を有する．多くの学習者がソースコード作成過程に陥りやすい間違いをパターン化して指導者が把握できていれば，学習者が行き詰る内容を把握するまでの時間が削減でき，指導効率が向上する．ソースコード作成過程を把握するための研究として文献 [1], [2], [3] が挙げられる．しかし，これらの手法を用いて学習者が頻繁に陥る間違いをパターン化することは難しい．

本論文では，多くの学習者に共通して見られる関数の呼び出しパターンを学習パターンとして導出する手法を提案する．本手法により，指導者は多くの学習者に共通して現れる学習パターンを把握できる．発見された学習パターンから，間違った学習パターンを探し，それらに基づいて事前に指導方針を用意できるので，指導効率を向上させられる．

2. プログラミング教育における関数呼び出し

2.1 プログラミング演習における問題点

プログラミング演習では，学習者は与えられた課題に個人で取り組み，分からない内容を指導者に質問する．学習者は，プログラム作成過程で頻繁に分からない内容に直面し行き詰まる．行き詰った学習者は，何度もソースコードを変更し実行などの試行錯誤を繰り返し，行き詰まりの原因を発見し，自己解決しようとする．ここで自己解決できない学習者は，指導者の助けを必要とする．

指導者は質問を受けたさいに，学習者が理解に苦しんでいる内容を把握しなければならない．しかし行き詰った学習者は，自身の行き詰まりの原因を指導者に説明できない場合がある．そのため，指導者は学習者が作成したソースコードをよく読み返したり問答をしたりする必要がある．これにより，学習者一人への対応に時間を費やし，学習者全員への指導が不十分になってしまう．

現在では，質問を受けた指導者は学習者の理解度や行き詰る内容をソースコードをもとに判断している．しかしソースコードは，学習者の演習の結果である．よってソースコードから学習者が試行錯誤したであろう学習過程を判断することは難しい．学習者が理解に苦しんだ内容を把握するには，演習の過程に着目する必要がある．

大学のプログラミング演習では，毎年同じような演習課題を出題することが多い．同じような課題について，演習途中で理解に苦しむ内容は多くの学習者で共通しており，試行錯誤する内容も多くの学習者で共通した特徴があると考えられる．しかし現状では，学習過程を考慮できておらず，これらを把握できていない．効率的に適切な指導を与えるためには，演習の事前に行き詰ると考えられるパターンを過去のプログラミング演習時の記録から抽出し，そのパターンと比較することにより行き詰まりの内容を即時に特定する必要がある．

2.2 既存研究

演習過程に得られる記録から学習状況を分析する研究として文献 [4], [5] が挙げられる．文献 [4] の手法では，プログラムの行数，実行回数，変数や構文の数の遷移を記録することで学習者の進捗を管理できる．しかしこの手法では，学習者が行き詰っている内容まで分析できない．文献 [5] の手法では，演習過程で発生したエラーメッセージの解析により学習者の特徴を分析できる．しかし演習過程に発生するコンパイルエラーを解析の手がかりとしているため，エラーは出ないが採用したアルゴリズムの間違いが原因の行き詰まりを発見できない．

プログラム作成過程の分析は，プログラム理解の分野で行われている [6]．文献 [6] では Eclipse を用いたプログラムの開発・保守におけるプログラム作成過程の分析をしている．プログラム作成過程として，ソースコードに影響を与えるすべての操作履歴を記録する．これらの操作履歴からソースコードの差分を抽出し，開発過程を追跡する．この手法をプログラミング演習で用いた場合プログラム作成過程を再現できるが，多くの学習者が理解に苦しむ内容を典型的なパターンとして絞り込むことが困難である．

初期のプログラミング教育でよく用いられる手続き型言語を用いたプログラムは，順次・選択・反復の 3 つの制御と関数呼び出しで記述できる．学習者はソースコードを作成するために，3 つの制御を用いて関数呼び出しを行っているため，関数の呼び出しは，学習者の作成するプログラムに依存する．したがって関数呼び出しの履歴は，学習者のプログラムの特徴を表している．関数呼び出しの履歴を分析すれば，多くの学習者が陥る間違いを典型的なパターンとして把握できる．

3. 関数呼び出しの記録と学習パターンの導出

3.1 プログラムの作成過程の記録と分析

本論文では，学習者のプログラム作成過程を収集し，機械的に分析することにより，学習者のプログラム作成過程をパターン化する手法を提案する．本手法では，学習者のプログラム作成過程をとして関数呼び出しの履歴

†立命館大学大学院 理工学研究科
‡立命館大学 情報理工学部

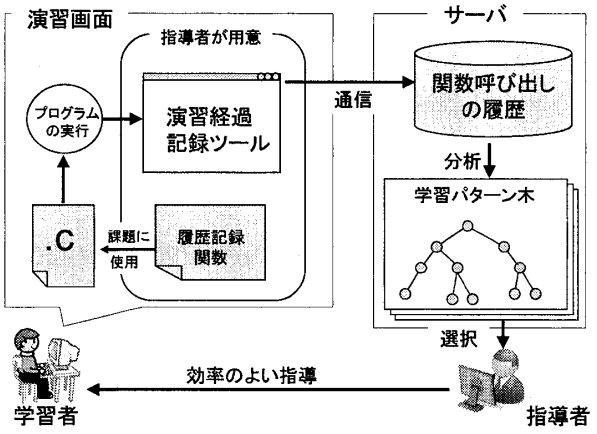

図 1: 手法の流れ

を記録する．さらに木構造を用いて記録された関数呼び出しの履歴を分岐の程度に応じて可視化する．その中から指導者が適切なものを選択することにより，学習者が理解に苦しむ内容を表す関数呼び出しのパターンを学習パターンとして導出する．本手法の流れを図 1 に示す．本手法では，プログラムの作成過程を記録するために使用する履歴記録関数とプログラムの実行結果を出力しその経過を記録する演習経過記録ツールを用いる．まず指導者は，履歴記録関数と演習経過記録ツールとそれらを用いたプログラム作成の課題の 3 つを用意する．学習者は，これらを用いてプログラム作成の演習をする．この演習から得られた関数呼び出しの履歴を分析することで，学習者に共通して現れる学習パターンを導出できる．本手法により，出題した課題についての学習者が陥りやすい間違いパターンを把握できる．これにより，再び同じ課題を用いた演習をするさいに，事前に指導方針を用意することで，指導の効率を向上させられる．

3.2 関数呼び出しの履歴の収集

学習者は履歴記録関数と演習経過記録ツールを用いた演習をする．履歴記録関数が呼び出されると，演習経過記録ツール上に実行結果が出力される．出題する演習課題は，3 つの制御と履歴記録関数を用いて達成できる．学習者は，指導者が示す実行結果と同一の出力をするプログラムを作成することを目的に演習をする．履歴記録関数の呼び出しに応じて，実行結果が表示され，呼び出した履歴記録関数の呼び出し履歴が記録される．関数呼び出しの履歴には，呼び出された履歴記録関数の種類とそのとき履歴記録関数に与えられた引数が記録される．

3.3 学習パターンの導出

学習者から収集された関数呼び出しの履歴が分析され，多くの学習者に共通する関数呼び出しのパターンの候補が学習パターン木という木構造として可視化される．指導者は可視化された学習パターン木の中から適切なものを選択する．指導者は選択した学習パターン木のノードに着目し，学習者がつまずく要因となる学習パターンを導出する．学習パターン木の分岐点となるノードは，学習者に共通していた関数呼び出しのパターンが分岐したことを表している．このノードは，複数の学習者で同一のことをしなくなった箇所であり，学習者の考え方が分

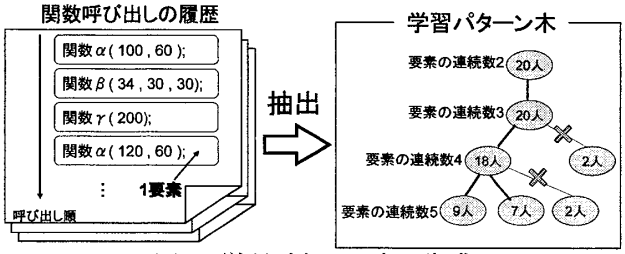

図 2: 学習パターン木の生成

岐した箇所と考えられ，つまずく要因を表していると考えられる．よって本研究では，分岐が多く，かつ，子ノードでの学習者が分散しているノードに着目する．

学習パターン木を生成する方法を図 2 に示す．関数呼び出しの履歴を構成する履歴記録関数の種類と呼び出し時に履歴記録関数に与えた引数の組 1 つ分を 1 要素とする．関数呼び出しの履歴の中の連続する要素の並びに学習者のプログラム作成過程が現れている．この連続した要素を呼び出しパターンと呼ぶ．呼び出しパターンに含まれる連続して現れる要素の個数を用いて，要素の連続数と呼ぶ．次の手順で学習パターン木が生成される．

1. 全学習者の関数呼び出し履歴から最頻出する要素の連続数が 2 の呼び出しパターンを見つけ出す
2. 現在までに発見された要素の連続数が n 個の呼び出しパターンからはじまる要素の連続数が (n+1) 個の呼び出しパターンを探す
3. 手順 2 を再帰的に繰り返し，要素の連続数が大きい呼び出しパターンを見つける．見つかった呼び出しパターンを共通して呼び出した学習者が一定数 α 未満になるまで繰り返す．このとき，共通して呼び出された学習者が一定数未満の要素は学習パターン木から取り除く

4. 図を描画する演習での実験

4.1 演習内容

本手法を用いて学習パターンを導出できるかを検証するために，本手法を用いたプログラム作成の演習をし，関数呼び出し履歴を収集した．関数呼び出しの履歴を収集するために，図を描くための履歴記録関数と実行結果の図を表示する演習経過記録ツールを用いた演習をした．演習経過記録ツール上には，指導者が用意した履歴記録関数の呼び出しに応じて図が描画される．学習者は 3 つの制御と履歴記録関数を用いた C 言語によるプログラム作成の演習をする．この演習を大学生 17 名を対象に行った．演習課題として図 3 に示す図を与えた．

4.2 履歴記録関数と演習経過記録ツール

今回の演習で用いた履歴記録関数は C 言語の関数として用意された関数である．履歴記録関数には，直線の描画，色の変更，太さの変更をするためのものがある．この関数が呼び出されるたびに図形描画に関する命令が演習経過記録ツールに送信される．演習経過記録ツールは履歴記録関数からの描画命令に応じて，図を描画する．今回行った演習の実行例を図 4 に示す．図 4 のソースコード中の①,②,③で示された関数が今回演習で用いた履歴記録関数である．①は直線の太さを変える関数であり，直線の太さが 10px に設定される．②は直線の色を変える関数であり，直線の色が赤色に設定される．③は直線

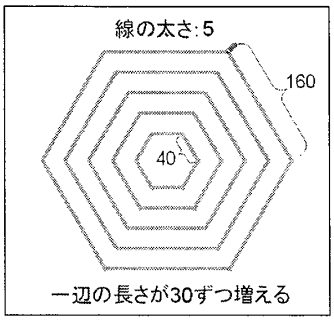

図 3: 演習課題の図

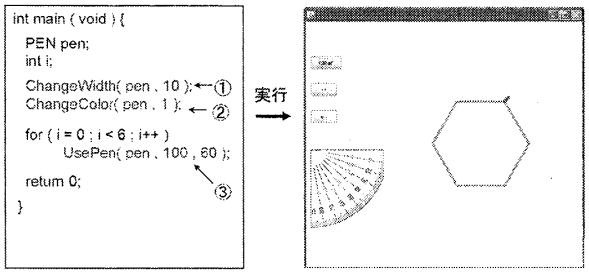

図 4: 履歴記録関数を用いたソースコード作成の演習

を描画する関数であり，100px の直線が 60 ° の方向に描画される．③の関数が 6 回連続で呼び出され正六角形が演習経過記録ツール上に描画される．

5. 評価

5.1 学習パターン木の生成

前節で述べた演習から収集された関数呼び出しの履歴から学習パターン木を生成した．共通する学習者数が少なかった呼び出しパターンを学習パターン木から取り除くさいの学習者数を閾値 α とする．共通する学習者数が α 人未満のノードを学習パターン木から取り除く．

α =1 の場合，すべての学習者の呼び出しパターンが学習パターン木に現れる．そのため，ある一人のみによる関数呼び出しの結果も学習パターン木に反映される．よって生成され表示される学習パターン木のノードの数が膨大な量になる．また α =3 の場合，生成される学習パターン木のノードが少なく，発見できる学習パターンが少ないので，間違いの要因となる学習パターンを見つけられなくなる．今回は，学習パターン木のノード数が適度であると考えられる α =2 の場合について分析する．

5.2 間違いを表す学習パターン

閾値 α =2 の場合の学習パターン木の一部を図 5 に示す．図 5 の学習パターン木を構成する各ノードには，学習者に共通して現れた呼び出しパターンが含まれている．ノード上の数字は，そのノードに含まれている呼び出しパターンを共通して呼び出した学習者の数である．学習パターン木の根からノード (a) まで分岐は現れなかった．これはノード (a) に含まれる要素の連続数が 7 の呼び出しパターンは多くの学習者に共通しており，この呼び出しパターンに至るまでに違ったパターンの関数呼び出しが見られた学習者は少なかったことを表している．

分岐となるノードに着目し，学習者の間違いを表して

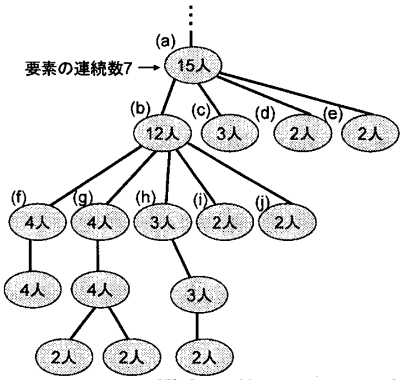

図 5: α =2 の場合の学習パターン木

いる学習パターンを探すと，学習パターン木の根に一番近い分岐点としてノード (a) が見つけられる．ノード (a) に含まれる呼び出しパターンから派生する要素の連続数が 1 つ多い呼び出しパターンとしてノード (b), (c), (d), (e) が見つけられる．ノード (b) に含まれる呼び出しパターンは正解パターンの一部を表すものであった．一方ノード (c), (d), (e) に含まれる呼び出しパターンは正解とも間違いとも判断がつかないものだった．

次の分岐点としてノード (b) が見つけられる．ノード (b) に含まれる呼び出しパターンから派生する要素の連続数がひとつ多い呼び出しパターンとしてノード (f), (g), (h), (i), (j) が見つけられる．ノード (f), (g), (h) に含まれる呼び出しパターンは正解パターンの一部を表すものであった．一方ノード (i), (j) に含まれる呼び出しパターンは間違いを表しているものであった．

分岐点となるノード (a) とノード (b) では，ノード (b) に含まれる呼び出しパターンの方がより起こりやすい間違いを表していた．ノード (a) の子ノードである (b), (c), (d), (e) に含まれる呼び出しパターンを共通して呼び出した学習者の数はそれぞれ 12 人，3 人，2 人，2 人であった．これより，ノード (a) に含まれる呼び出しパターンが見られた学習者の多くから，(b) に含まれる呼び出しパターンが見られることが分かる．一方ノード (b) の子ノードである (f), (g), (h), (i), (j) に含まれる呼び出しパターンを共通して呼び出した学習者の数はそれぞれ 4 人，4 人，3 人，2 人，2 人であった．ノード (b) に含まれるの呼び出しパターンが見られた学習者が次に呼び出す呼び出しパターンは，ノード (a) の分岐よりもばらつきがあることが分かる．以上の結果から，子ノードに含まれる呼び出しパターンを共通して呼び出した学習者の数が分散している分岐点が，より学習者の間違いを表しているといえる．ノード (b) から分岐した呼び出しパターン (f), (g), (h), (i), (j) をそれぞれ本手法により導出された学習パターン F, G, H, I, J とする．

5.3 学習パターンの分類

今回の演習では，実行結果が図として表示されるため，実行結果の図の見た目から判断して学習パターンを分類した．実行結果の見た目より分類された学習パターンは，正解のものを含め図 6 に示す 3 パターンに分類できた．これらを外観の分類による学習パターン X,Y,Z とする．導出された学習パターン F,G,H,I,J と外観の分類による学習パターン X,Y,Z との比較を表 1 に示す．

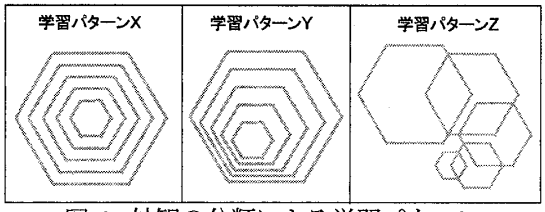

図 6: 外観の分類による学習パターン

表 1: 学習パターンの分類

本手法により導出された学習パターン	外観の分類による学習パターン
F	X
G	X
H	X
I	Y
J	Y

本手法により導出された学習パターン F,G,H,I,J は，外観の分類による学習パターン X,Y,Z のうち X,Y の 2 つに分類された．本手法により導出された学習パターン F,G,H はそれぞれ別のパターンとして導出されたが，そのすべてが外観の分類による学習パターンでは X に分類できた．本手法により導出された学習パターン I,J についても同様に，どちらとも外観の分類による学習パターンでは Y に分類できた．また，外観の分類による学習パターン Z は本手法からは導出されなかった．

5.4 考察

本手法により導出された学習パターン F,G,H は，すべて外観の分類による学習パターン X に分類された．本手法により導出された F,G,H は，実行結果の図としてはすべて同じような図をしているが，呼び出しパターンを構成する要素が異なるものであることを表している．本手法では，実行結果の見た目では判別のつかない要素の違いで学習のパターンを導出できていることが分かる．

外観に分類による学習パターン Z は，本手法では導出されなかった．この理由として，外観の分類による学習パターン Z は実行結果では同じような図に見えるが，呼び出しパターンを構成する要素が異なっているものが何通りかあったと考えられる．さらに，呼び出しパターン木に現れていないため，今回の場合，共通して呼び出した学習者が 2 人以上のものがなかったと考えられる．学習パターン木を閾値 α=1 で生成したとき，外観の分類による学習パターン Z は共通して呼び出した学習者が 1 人の呼び出しパターンとしていくつも発見された．このことから，外観の分類による学習パターン Z は，実行結果の見た目には同じような間違いをしているように見えるが，その原因はさまざまでありパターン化できないことが分かった．このような誤りをしている学習者を指導するには，外観の分類からだけでは原因が特定できない．外観の分類だけからの指導は，異なる原因を想定したものになる可能性があり，指導がかえって学習者の混乱を引き起こす可能性すらあるといえる．

以上のことから，図 7 に示すように本手法で導出された学習パターンと外観の分類による学習パターンでは，原因の特定という観点で異なる粒度のものが発見されていることがわかる．本手法により導出された学習パターンは，実行結果の外観の分類による学習パターンよりも原因をより細かく限定できる．本手法により，指導者は

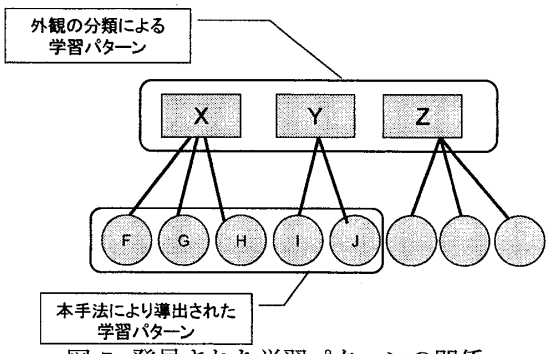

図 7: 発見された学習パターンの関係

多くの学習者に共通する間違いのパターンを用いて，間違いの原因を正確に把握でき，その間違いの原因となる関数呼び出しを調べられる．それにより，事前に間違いのパターンに応じて対策を講じられる．

6. おわりに

本論文では，関数呼び出しの履歴を収集し分析することで，多くの学習者に共通する関数呼び出しのパターンを学習パターンとして導出する手法を提案した．関数呼び出しの履歴に着目した分析により，少ない履歴の分析で出題した課題に対して多くの学習者に共通する間違った学習パターンとその原因となる関数呼び出しを把握できる．発見された間違った学習パターンに応じて事前に指導方針を講じ対策をとれるので，次に同じ課題での演習をするさいに，指導者は学習者ひとりへの指導にかかる時間を削減でき，指導効率を向上させられる．

本手法を用いて，17 名の大学生を対象に演習を実施した．この演習より得られた関数呼び出しの履歴から，間違いの学習パターンを導出したところ，演習中に見られた学習パターンよりも詳細に学習パターンを抽出できた．

今後は，どのようなプログラミング課題に対して本手法が有効であるかを検証する予定である．

参考文献

[1] Miryung K., David N., Program element matching for multi-version program analyses, International Conference on Software Engineering, pp.58–64, 2006

[2] Iulian Neamtiu, Jeffrey S. Foster, Michael Hicks, Understanding source code evolution using abstract syntax tree matching, International Conference on Software Engineering, pp.1–5, 2005

[3] 水穂，土田，浜名，佐藤，PSF 手法に基づくプログラム作成過程における変遷の可視化，Journal Article, Vol.105, No.332, pp.37–42, 2005

[4] 張，西田，安倍，石橋，松浦，プログラミング授業を支援する学習環境 PEN＋，Journal of Informatics, Vol.5 No.1, 2008

[5] 西，劉，横田，デバッガとの連携による C 言語学習支援システムの提案，電子情報通信学会技術研究報告，Vol.106, No.583, pp.173–178, 2007

[6] 大森，丸山，開発者による編集操作に基づくソースコード変更抽出，情報処理学会論文誌，Vol.49, No.7, pp.1234–1244, 2008