プログラミング学習におけるセグメント分割を用いたソースコードの誤り箇所特定方法の提案

2018SE078 澤田侑希　2018SE096 梅田祐一郎
指導教員：蜂巣吉成

1　はじめに

C 言語のプログラミング演習で，学習者は与えられた演習問題に取り組む．その中で，コンパイラが検出不可能な誤りは，学習者が自力で誤り箇所を特定する必要がある．しかし，プログラミングの知識が少ない学習者にとって誤り箇所の特定は難しい場合がある．演習問題は比較的規模が小さいプログラムであるので，学習者が間違える箇所は限られる．事前に模範解答とフィードバックメッセージを作成することで，自動フィードバックシステムとして学習者を支援できる可能性がある．それを実現するためには，フィードバックに適した誤り箇所の特定が必要である．学習者が解答したプログラム (以下，修正対象プログラム) と模範解答のプログラム (以下，模範解答プログラム) を比較して差分を生成し，学習者のソースコードの誤り箇所を特定することができる．

一般に，模範解答には別解が存在する．修正対象プログラムとすべての模範解答プログラムを比較することで，正確な誤り箇所を特定できる．しかし，想定されるすべての別解を模範解答プログラムとして用意する必要があるという問題点がある．

本研究では，用意する模範解答プログラムの総数を削減するために修正対象プログラムと模範解答プログラムをセグメント単位で比較し，学習者に適した自動フィードバックシステムの実現に向けたソースコードの誤り箇所特定方法を提案する．本研究におけるセグメントとは，プログラムを分岐や合流のない文の列に分割したものである．セグメントにより，プログラムを一定の意味を持つコード片の集合として捉えることができる．セグメントごとに別解を作成し，それらを組み合わせて数多くの模範解答プログラムの別解を網羅することができる．すなわち，用意する模範解答プログラムの総数と，模範解答についてプログラム単位で別解を考える負担を削減できる．

提案方法を評価するために，誤り箇所を特定するツールを実装した．作成したツールを用いて 10 個のプログラムで誤り箇所の特定に成功した．誤りがあるセグメントが 2 個のプログラムでは，約 1 時間で誤り箇所を特定することができた．

2　関連研究

肥後らの研究では，設定ファイルを考慮した自動バグ限局手法を提案している [1]．テストケースによる実行経路の情報を用いて自動バグ限局を行う手法“Spectrum-Based Fault Localization”[2] を用いて，ソースコード以外のファイルに対するバグの原因箇所である可能性の計算を行っている．テストで実行するファイルのみならず，プロパティファイルに対しても計算を適用することで，バグが含まれる可能性のある調査対象を拡張している．しかし，手法を利用するために複数のテストケースが必要になる．

Shalini らの研究では，半教師あり検証済みフィードバック生成方法を提案している [3]．学習者のプログラムをクラスタ化し，誤りがあるプログラムに対して類似度が高く正しい実行結果を得るプログラムを対応付ける．しかし，別解が多い問題ではクラスタが増加し，そこに正しい実行結果が得られるプログラムが存在しない場合，手動で作成して再度クラスタリングする必要がある．別解が多い問題に弱い．

3　誤り箇所特定方法の提案

3.1　誤り箇所特定方法の概略

プログラムをコード片の集合と捉え，コード片の組み合わせにより模範解答の別解を網羅する．修正対象プログラムと模範解答プログラムをコード片単位で比較し誤り箇所を特定する．模範解答の別解は一定の規則に従いコード片単位で作成する．修正対象プログラムも同様の規則でコード片に分割し，作成した模範解答のコード片と差分を取る．コンパイラによる検出が不可能な誤りは数値や演算子などといった比較的少数のトークンの記述ミスであるという考えの元，差分が小さい模範解答のコード片を修正対象プログラムのコード片と置き換えて実行し，プログラムが正しく動作したならばそのコード片が誤り箇所だと判断する．

3.2　セグメントの定義

プログラムを一定の意味を持つコード片の集合と捉えるために，プログラム文の分岐や合流の境目を基準としてコード片に分割し，これを「セグメント」と定義する．セグメント分割には制御フローグラフにおける基本ブロックの考え方を用いる [4]．C 言語における制御文については予約語と括弧で囲まれた式を 1 個のセグメントとする．ただし，フィードバックの対象とする関数の関数名と波括弧，制御構造の波括弧はセグメントに含めないものとする．

3.3　セグメント分割の効果

修正対象プログラムの誤り箇所を特定するとき，模範解答プログラムと比較することで特定できるが，想定されるすべての別解を模範解答プログラムとして用意する必要がある．しかし，模範解答をセグメント単位で作成することで，教員が別解をプログラム単位で作成する負担を削減す

る効果が得られる．例えば，ソースコード 1 は if 文と for 文を用いて累乗を計算するプログラムである．

ソースコード 1　プログラム例 1

```
1  double func1(double p1, int p2)
2  {
3    int v1;
4    double v2;
5    v2 = 1;
6    if (0 <= p2) {
7      for (v1 = 0; v1 < p2; v1 = v1 + 1) {
8        v2 = v2 * p1;
9      }
10   } else {
11     for (v1 = 0; p2 < v1; v1 = v1 - 1) {
12       v2 = v2 / p1;
13     }
14   }
15   return v2;
16 }
```

ソースコード 1 は，以下の 8 つのセグメントに分割する．

1. int v1; double v2; v2 = 1;
2. if (0 <= p2)
3. for (v1 = 0; v1 < p2; v1 =v1 + 1)
4. v2 = v2 * p1;
5. else
6. for (v1 = 0; p2 < v1; v1 =v1 - 1)
7. v2 = v2 / p1;
8. return v2;

3 個目と 6 個目のセグメントは while 文へと書き換えることができる．また，for 文の繰返しの範囲や if 文の条件分岐の順番など書き換えられる要素が多く存在する．それらを組み合わせることで，ソースコード 1 のような難易度の低い問題でも，想定される別解のプログラム総数は 1,000 個を超える．しかし，セグメント単位で模範解答を作成することで，模範解答の総数は約 80 個のセグメントに削減できる．

3.4　提案方法

本研究では，修正対象プログラムをセグメント (以下，修正対象セグメント) に分割し，模範解答のセグメント (以下，模範解答セグメント) との差分を用いて誤り箇所を特定する方法を提案する．各修正対象セグメントと模範解答セグメントを比較することで，誤りが存在するセグメント (以下，誤りセグメント) を特定する．

誤りセグメントを特定することで，ソースコードの誤りが全体のどこに存在するか範囲を絞ることができる．フィードバックを行うためには詳細に誤り箇所を特定する必要があるので，セグメントをトークン単位で比較し，誤り箇所をトークン単位で特定する．トークン単位の差分を用いて誤り箇所の特定が可能であるので，差分を情報 (以下，差分情報) として生成して出力する．

前提条件

本研究における修正対象プログラムは，次の前提条件を満たすものとする．

1. コンパイルは可能だが，正しい実行結果を得られない．
2. 学習者は指定された関数を記述する．
3. 解答する関数内の変数宣言は先頭で行う．
4. 変数の変数名と使い方に制限を設ける．
5. 計算式では余分な記述をしない．

誤り箇所を特定した差分情報を出力する処理の流れを図 1 に示す．また，誤り箇所を特定した差分情報を出力する過程は 8 段階である．

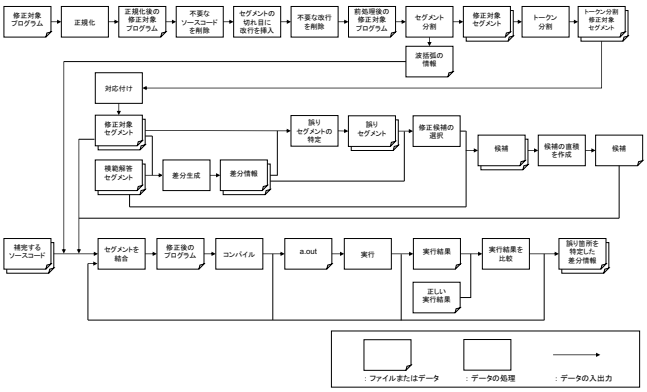

図 1　提案方法の処理の流れ

3.4.1　正規化

学習者が記述し得るソースコードを網羅した模範解答セグメントの作成は制約を設けない限り困難である．本研究では修正対象プログラムに対して正規化の処理を行うことで，模範解答セグメントを削減して作成可能にする．正規化の処理は次の通りである．

1. 余分な空白文字，改行文字，コメントを削除する．
2. 変数宣言と初期化式がまとめて記述されている場合，別々に書き換える．
3. 不等号の向きを>から<，>=から<=に書き換える．
4. 計算式の形式が単項演算の場合，二項演算に換える．
5. 制御構造の波括弧が省略された場合，波括弧を付ける．

3.4.2　セグメント分割

セグメント分割の前処理として，セグメントの境目になり得る箇所のみに改行文字を挿入する．続いて，改行文字の前後でソースコードを分割し，セグメント候補とする．

次に，文字列を走査してセグメント候補がセグメントの定義通りの構成である場合，セグメントとする．セグメント候補の文字列が改行文字のみの場合，セグメントの対象

外とする．

3.4.3 トークン分割

修正対象セグメントと模範解答セグメントをトークン単位で比較するために，修正対象セグメントのソースコードをトークンごとに改行し，トークン分割をする．

3.4.4 差分情報の生成

修正対象セグメントと模範解答セグメントを対応付け，差分情報を生成する．差分情報は，修正対象セグメントと模範解答セグメントの差分を検出箇所とともにまとめたものである．修正対象セグメントと模範解答セグメントが一致する場合，差分ゼロが差分情報として出力される．

模範解答セグメントは個別に作成するので，プログラムにおける何の役割のセグメントであるかわからない．修正対象セグメントはすべての模範解答セグメントと総当たりで差分情報を生成する．

3.4.5 修正するセグメントの選択

差分情報を用いて，修正対象セグメントから誤りセグメントを特定する．差分ゼロの差分情報が存在しない修正対象セグメントを誤りセグメントとして選択する．また，差分ゼロの差分情報が存在する場合でも，誤りセグメントと模範解答セグメントが誤って対応付いている可能性がある．この場合，修正対象セグメントを順に誤りセグメントの可能性があるものとして修正を試みる．

3.4.6 セグメントの修正候補の選択

生成した差分情報で誤り箇所が特定できるか確認するために，誤りセグメントを修正したプログラムを作成してテストする．誤りセグメントを修正するために，セグメントの修正候補 (以下，候補) を選択する．修正は修正対象セグメントと模範解答セグメントの置換で実現する．差分情報の差分の大きさから候補とする模範解答セグメントを決める．

差分が小さい差分情報は，学習者の意図に近い模範解答セグメントから生成された差分情報である可能性が高い．差分情報の差分の大きさをしきい値として，しきい値より小さい差分情報の生成元の模範解答セグメントを候補とする．候補は，差分情報の差分が小さい順に並び替える．

3.4.7 候補の直積の作成

誤りセグメントが複数存在する場合，修正したプログラムを作成するために，誤りセグメントの組み合わせを考慮する必要がある．この場合は候補の直積を作成し，これを候補とする．どのセグメントが誤りセグメントかわからない場合，作成したすべての候補の直積を結合したものを候補とする．

3.4.8 修正したプログラムのテスト実行

候補を用いて誤りセグメントを修正する．セグメントを結合し，セグメント分割の前処理で削除した関数や波括弧などのソースコードを補完してプログラムを作成する．

作成したプログラムをコンパイルおよび実行する．コンパイルエラーが発生した場合,「コンパイルエラー」を出力して次の候補の処理を行う．無限ループが発生した場合，タイムアウトさせて「タイムアウト」を出力し，次の候補の処理を行う．続いて，生成したプログラムの実行結果と正しい実行結果を比較する．一致した場合,「成功」を出力して各セグメントの誤りの有無を出力する．また，誤りセグメントでは差分情報を出力する．不一致の場合,「失敗」を出力して次の候補の処理を行う．

すべての候補のプログラムの実行結果を比較して一致しなかった場合，どのセグメントが誤りセグメントかわからない場合の誤りセグメントが他にも存在する可能性がある．この場合，特定の対象とするセグメントを増やして再度候補を選択し，プログラムを作成し，実行結果を比較する．

3.5 誤り箇所特定に要する時間

作成したプログラムをテスト実行する処理は，膨大な時間を要する場合がある．候補は，誤りセグメントが 1 個増えると，セグメントの組み合わせを考慮する必要があるため大幅に増加する．プログラムを作成する処理の計算量を一般化し，誤りセグメントの数ごとの実行時間の目安を計算する．フィードバックシステムに実用的なツールを作成するために，対象とする誤りセグメントの最大数を検討する．なお，1 つのセグメントに誤り箇所が複数あっても，誤りセグメントは 1 つとして数える．

3.5.1 計算量の分析

模範解答セグメントの総数を m とすると，修正対象セグメントごとに m 個の差分情報を生成する．誤りセグメントの数を k とすると，m^k 個のプログラムを作成する必要がある．また，修正対象セグメントの総数を n とすると，どのセグメントが誤りセグメントかわからない場合，誤りセグメントの選び方は ${}_n\mathrm{C}_k$ 通りである．この場合，${}_n\mathrm{C}_k \times m^k$ 個のプログラムを作成する必要があり，最悪の計算量となる．

3.5.2 誤りセグメントの最大数

修正対象セグメントが 10 個，模範解答セグメントが 100 個で，プログラムの作成と実行に 0.1 秒要する場合の実行時間を，誤りセグメントの数ごとに計算する．

誤りセグメントが 1 個の場合，計算量は ${}_{10}\mathrm{C}_1 \times 100^1$ となり，実行時間は 1 分 40 秒である．また，誤りセグメントが 2 個の場合，計算量は ${}_{10}\mathrm{C}_2 \times 100^2$ となり，実行時間は 12 時間 30 分である．また，誤りセグメントが 3 個の場合，計算量は ${}_{10}\mathrm{C}_3 \times 100^3$ となり，実行時間は 416 日と 16 時間である．

しきい値を用いて候補を削減するので，実行時間は計算結果より短くなる．誤りセグメントが 2 個までの場合，フィードバックシステムに実用的な時間内に誤り箇所が特

定できる可能性がある．本研究では，誤りセグメントを 2 個までに制限してツールを作成する．

4 ツールの適用例

本研究では，C 言語で累乗を計算するプログラムを作成する演習問題を対象として誤り箇所特定ツールを作成した．修正対象セグメントは最大で 16 個を想定し，模範解答セグメントは 87 個作成した．

誤りセグメントの数と誤りセグメントが特定可能かどうかで 5 種類に分類し，プログラムを 2 個ずつ作成した．作成した 10 個のプログラムをツールに入力し，誤り箇所が特定できたかという定性的な観点と，実行時間という定量的な観点で評価する．

4.1 実行環境

ツールは MacBook Pro (13-inch, M1, 2020) で実行した．CPU は 8 コアで，メモリは 16GB，OS は macOS Monterey 12.1 である．ただし，ツールでは並列処理モジュールは使用していない．また，しきい値は 10 で，タイムアウトは 1 秒で発生するように設定した．実行時間は誤差が生じるので，3 回実行して平均した．

4.2 実行結果と評価

作成した 10 個のプログラムでは，すべてのプログラムにおいて誤り箇所の特定に成功した．定性的な観点において提案方法は妥当である．しかし，設定するしきい値が誤り箇所を特定した差分情報より小さい場合，誤り箇所の特定に失敗すると考えられる．

作成した 10 個のプログラムにおいて，実行に最も時間を要したプログラムは，修正対象セグメントが 16 個で，誤りセグメントが 2 個存在し，いずれも誤りセグメントが特定できないものであった．実行時間は 1 時間 3 分 19 秒であった．分岐の制御構造の入れ子のセグメントでは，5.2 節で述べるような対応付けを行っているので，生成する差分情報の総数を削減できた．また，しきい値を設けて候補を削減したので，作成するプログラムを削減できた．

5 考察

5.1 実行時間

4.2 節の実行結果より，誤りセグメントを 2 個までに制限した場合，フィードバックシステムに実用的な時間内に誤り箇所が特定できる可能性が高い．この場合，即時のフィードバックは困難であるが，フィードバックシステムは授業時間外の演習課題などの状況で使用できる．

5.2 生成する差分情報の削減

模範解答セグメントを処理ごとに分割して作成し，同様の処理をするセグメントを対応付けて比較することで，生成する差分情報を削減できる．一部のセグメントでは，修正対象セグメントを走査することで対応付けが実現できる．データフロー解析などを利用してセグメントの依存関係を考慮することで，すべてのセグメントで対応付けが実現できる可能性がある．

5.3 自動フィードバックシステムの実現

差分情報からフィードバックメッセージを選択する方法を検討する．差分情報には，修正対象プログラムにおける不要なソースコードと不足しているソースコードが記述されている．文字列とフィードバックメッセージを対応付けたデータを作成し，差分情報の文字列とパターンマッチすることで，適切なフィードバックメッセージを選択できる可能性がある．

5.4 他言語の拡張性

本研究では，C 言語を対象として誤り箇所特定方法を提案した．しかし，C 言語以外のプログラミング言語であっても，提案方法で誤り箇所の特定ができる可能性がある．

例として，Python に提案方法を適用する場合について検討する．Python では，インデントを用いて制御構造を実現している．そのため，制御文とインデントを基準に分割することで，セグメント分割をすることができる．したがって，提案方法を Python に適用できる可能性が高い．

6 おわりに

本研究では，セグメント分割を用いてプログラミング学習におけるソースコードの誤り箇所の特定方法を提案した．

今後の課題として，依存関係を考慮したセグメントの対応付けの実現，自動フィードバックシステムの実現が挙げられる．また，しきい値とツールの性能の関係性，他言語における提案方法の実現可能性を調査する必要がある．

参考文献

[1] 肥後芳樹，まつ本真佑，内藤圭吾，谷門照斗，楠本真二，切貫弘之，倉林利行，丹野治門：設定ファイルを考慮した自動バグ限局手法の拡張，ソフトウェアエンジニアリングシンポジウム 2019 論文集，vol.2019，pp.97-105 (2019).

[2] W. Eric Wong，Ruizhi Gao，Yihao Li，Rui Abreu，Franz Wotawa：A Survey on Software Fault Localization，IEEE Transactions on Software Engineering (TSE)，Vol.42，No.8，pp.707-740 (2016).

[3] Shalini Kaleeswaran，Anirudh Santhisr，AdityaKanade：Semi-supervised Verified Feedback Generation，FSE 2016：Proceedings of the 2016 24th ACMSIGSOFT International Symposium on Foundationsof Software Engineering (2016).

[4] 中田育男：コンパイラの構成と最適化，pp.282-284，朝倉書店 (2009).