プログラミング学習者の誤りプログラムと模範解答プログラムの実行履歴比較による誤り箇所推定方法の提案

M2023SE010 戸谷剛士
指導教員：蜂巣吉成

1 はじめに

プログラミング学習において，コンパイル可能であっても実行結果が期待通りにならないプログラムの誤り箇所を特定することは，プログラミング初学者にとっては難しい場合がある．誤りを特定するためには，デバッグ技術や論理的思考を用いて，プログラムの実行履歴を追跡し，どの部分で期待と異なる結果が生じているのかを必要があり，時間と労力を要する．

本研究では，模範解答プログラムと学習者が作成したプログラムの実行履歴を比較し，誤り箇所を推定する方法を提案する．模範解答プログラムと学習者のプログラムからそれぞれ実行履歴を取得し，得られた実行履歴を比較する．比較結果から誤りの可能性が高い文の疑惑値を計算し，学習者に示す．疑惑値は 0 から 1 の数値で表し，1 に近いほど誤りの可能性が高いことを示す．学習者は疑惑値の高い文から誤りか確認し，修正する．模範解答プログラムと学習者のプログラムは書き方や変数名が異なり単純には比較できない．ある結果を得るために必要な演算の回数や順序は基本的に同じであると考え，実行履歴から演算子を抽出し，条件判定，繰り返しなどの共通するものを抽象化する．抽象化された実行履歴を比較し，誤り箇所を推定する．

本稿では，C 言語プログラムを対象とし，次の 2 点を研究課題とする．

1. 実行履歴の抽象化方法
2. 疑惑値の計算方法

2 関連研究

Spectrum-Based Fault Localization (SBFL) とは，テストケースの実行トレースを利用した誤り箇所推定方法の 1 つである [1]．SBFL では，1 つのプログラムに対して複数のテストケースを実行し，成功したテストケースと失敗したテストケースにおいてどの文が実行されたかを調べ，文がフォールト箇所である可能性を疑惑値として定量化する．

Zheng らは学習者のプログラムの誤り箇所の特定のために，SBFL より誤り特定精度の高い Faulty Statement categorization Frequency-based Fault Localization(FSFFL) を提案している [2]．同じプログラムについて，学習者が誤る文は類似しているという結果から過去の学習者プログラムを用いて，プログラムの文を 7 つのカテゴリに分類し，それぞれのカテゴリで誤る頻度を求める．カテゴリの誤り頻度情報を用いて，従来の SBFL で計算された疑惑値に重み付けを行い誤り特定精度を向上させた．この手法は，カテゴリ別に誤る頻度を求めるために多くの学習者プログラムを必要とする．

本研究では 1 つのプログラムに対して複数のテストケースを実行する SBFL と違い，あるテストケースを模範解答プログラムに適用して実行した履歴を正しい実行履歴として，学習者の誤りプログラムの実行履歴を比較して，異なる箇所の情報から疑惑値を計算する．さらに疑惑値に対して FSFFL の考え方を参考に実行履歴の文に応じて疑惑値に重み付けを行い，誤り箇所推定の精度向上を図る．

3 実行履歴の比較による誤り箇所推定方法

3.1 誤り箇所推定の基本的なアイディア

3.1.1 概略

本研究の実行履歴の比較による誤り箇所推定方法の概略は次の通りである．

1. 模範解答プログラムと学習者が作成したプログラムの実行履歴を取得する
2. 取得した実行履歴を抽象化する
3. 抽象化した実行履歴を比較し疑惑値を計算する

実行履歴の比較の際に，`while` 文と `for` 文などの記述が異なるが同様の結果となるものを異なっていると判定しないように，本研究で扱うプログラムは `while` 文に統一して扱う．

3.1.2 実行履歴の取得方法

本研究における実行履歴は，C 言語プログラムの実行された文の並びである．これはデバッガのステップ実行などで取得できる．本研究ではデバッガを利用して作られた田中・戸田らのツール [3] を用いて取得する．

3.1.3 実行履歴の抽象化

模範解答プログラムと学習者プログラムの実行履歴を比較して誤り箇所を推定する必要があるが，制御文や変数名などが異なっており単純に比較できない．本研究では `sum += n % 10;`，`n /= 10;` などの数値計算の演算子を抽出し，`while (n > 0 || sum > 9)` などの制御文や `return 0;` を抽象化したものを抽象化した実行履歴とする．

実行履歴を抽象化する際に繰返しの条件やリテラルなど多くの情報を省いてしまうと，異なる箇所が少なくなり誤り箇所が実行履歴に含まれない可能性がある．多くの情報を残すと異なる箇所が多くなり誤り箇所を推定することが難しい．本研究では，実行履歴を次の 2 段階のレベルで抽象化を行う．

1. 抽象レベル低

   繰返し (演算, リテラル), 分岐 (演算, リテラル), 代入 (演算, リテラル)

2. 抽象レベル高

   繰返し, 分岐, 代入 (演算)

ソースコード 1, 2 の実行履歴と抽象化レベル低, 高で抽象化した結果を表 1 に示す. 抽象度低・高の各文を命令と呼ぶ.

表 1 デジタルルートを求める模範解答プログラムの実行履歴と抽象化した実行履歴の抜粋

	実行履歴	抽象化レベル低	抽象化レベル高
1	n = num;	代入=	代入=
2	while (n >= 10) {	繰り返し (>=10)	繰り返し
3	sum = 0;	代入=0	代入=
4	while (n > 0) {	繰り返し (>0)	繰り返し
5	sum += n % 10;	演算 +=%10	演算 +=%
6	n /= 10;	演算/=10	演算/=
7	while (n > 0) {	繰り返し (>0)	繰り返し
8	sum += n % 10;	演算 +=%10	演算 +=%
9	n /= 10;	演算/=10	演算/=
10	while (n > 0) {	繰り返し (>0)	繰り返し

3.1.4 抽象化した実行履歴の比較と疑惑値の計算

抽象化した実行履歴を diff コマンドで比較し, 異なる箇所に点数をつける. これを疑惑値と呼ぶ. 疑惑値は次のように求める.

1. 模範解答プログラムと学習者プログラムの異なる命令に疑惑値 S を付与する.

$$S = \frac{1}{F}$$

   F は模範解答プログラムと学習者プログラムの実行履歴において異なる命令の個数の合計である.

2. 実行履歴の各命令に対応する学習者プログラムと模範解答プログラムの文に疑惑値を累積する.
3. 累積した疑惑値の合計をその文の疑惑値とする.

3.1.5 誤り箇所推定の例

デジタルルートを求めるプログラムを例に説明する. デジタルルートを求めるプログラムをソースコード 1 に示す. 1999 のデジタルルートをソースコード 1 では 2 重ループを用いてデジタルルートを求めている.

ソースコード 1 デジタルルートを求めるプログラム (2 重ループ)

```
1  #include <stdio.h> //模範解答
2
3  int main() {
4      int num;
5      int n, sum;
6
7      // ユーザーからの入力を受け取る
8      // printf("数字を入力してください: ");
9      // scanf("%d", &num);
10     num = 1999;
11
12     // 数字の合計が一桁になるまで繰り返す
13     n = num;  // 変数の代入
14     while (n >= 10) { // 合計が一桁になるまで繰り返す
15         sum = 0;  // 合計の初期化
16         while (n > 0) { // 各桁を全て足すまで繰り返す
17             sum += n % 10; // 1の位の数字を取得して合計に加える
18             n /= 10;    // 1の位を取り除く
19         }
20         n = sum;    // 合計を次の計算対象にする
21     }
22     // デジタルルートを計算して表示
23     printf("%d のデジタルルート: %d\n", num, sum);
24
25     return 0;
26 }
```

ソースコード 1 を模範解答として, 誤り箇所の推定を行う. 誤りを含むプログラムをソースコード 2 に示す. ソースコード 2 では 1 重ループを用いてデジタルルートを求めるプログラムの合計変数の初期化 $sum = 1$;(14 行目) が誤りである.

ソースコード 2 誤りを含むデジタルルートを求めるプログラム 1

```
1  #include <stdio.h> //初期値の間違い
2
3  int main() {
4      int num;
5      int n, sum;
6
7      // ユーザーからの入力を受け取る
8      // printf("数字を入力してください: ");
9      // scanf("%d", &num);
10     num = 1999;
11
12     // 数字の合計が一桁になるまで繰り返す
13     n = num;
14     sum = 1; //sum = 0;
15     while (n > 0 || sum > 9) {
16         if (n == 0) {
17             n = sum;
18             sum = 0;
19         }
20         sum += n % 10;
21         n /= 10;
22     }
23     // デジタルルートを計算して表示
24     printf("%d のデジタルルート: %d\n", num, sum);
25
26     return 0;
27 }
```

実行履歴と抽象化レベル低, 高で抽象化した結果を表 2 に示す.

表 2 誤りを含むデジタルルートを求めるプログラム 1 の実行履歴と抽象化した実行履歴の抜粋

	実行履歴	抽象化レベル低	抽象化レベル高
1	n = num;	代入=	代入=
2	sum = 1;	代入=1	代入=
3	while (n > 0 || sum > 9) {	繰り返し (>0||> 9)	繰り返し
4	if (n == 0) {	条件分岐 (==0)	条件分岐
5	sum += n % 10;	演算 +=%10	演算 +=%
6	n /= 10;	演算/=10	演算/=
7	while (n > 0 || sum > 9) {	繰り返し (>0||> 9)	繰り返し
8	if (n == 0) {	条件分岐 (==0)	条件分岐
9	sum += n % 10;	演算 +=%10	演算 +=%
10	n /= 10;	演算/=10	演算/=

疑惑値の計算過程を図 1 に示す. 模範解答プログラムと誤りプログラムの抽象化レベル低で抽象化した実行履歴の異なる命令の合計数は 38 個である.

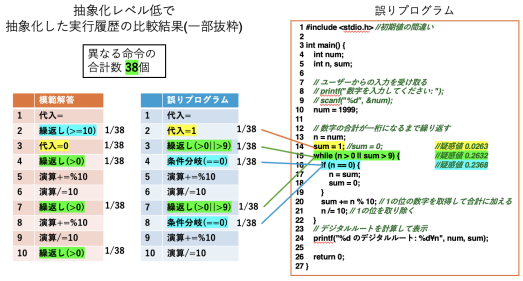

図 1　疑惑値の計算過程

抽象化レベル低で抽象化した実行履歴の比較結果より，異なる命令の合計数は 38 個であるので，異なる命令にそれぞれ 1/38 の疑惑値を付与する．付与した疑惑値を各命令に対応する文に累積し，その合計を疑惑値とする．

3.2　予備調査

2 重ループのデジタルルートを求めるプログラム (ソースコード 1) を模範解答プログラムとして，誤りを含むデジタルルートのプログラムを 8 つ用意し，それらのプログラムの疑惑値を計算する．誤りプログラムはソースコード 2 のように 1 重ループのデジタルルートを求めるプログラムの一部を誤りに変更して作成したもの 4 つと学習者が起こしうる編集途中のもの 4 つである．前者の例としてソースコード 2，後者の例として全桁の合計が二桁のときの配慮をしていない誤りプログラム 2 (ソースコード 1 の 15-19 行目のみ) に疑惑値を計算し，疑惑値の上位 5 位までをまとめたものを表 3，4，5，6 に示す．

表 3　デジタルルートの誤りプログラム 1(ソースコード 2) の疑惑値の上位 5 位 (抽象化レベル高)

疑惑値のついた文の説明				
順位	プログラムの種類	行数	文の内容	疑惑値
1	学習者プログラム	16	if (n == 0) {	0.5000
2	模範解答プログラム	14	while (n >= 10) {	0.2222
3	模範解答プログラム	20	n = sum;	0.1111
4	学習者プログラム	20	sum += n % 10;	0.0556
5	学習者プログラム	21	n / = 10;	0.0556

表 4　デジタルルートの誤りプログラム 1(ソースコード 2) の疑惑値の上位 5 位 (抽象化レベル低)

疑惑値のついた文の説明				
順位	プログラムの種類	行数	文の内容	疑惑値
1	模範解答プログラム	16	while (n > 0) {	0.2895
2	学習者プログラム	15	while (n > 0 || sum > 9) {	0.2632
3	学習者プログラム	16	if (n == 0) {	0.2368
4	模範解答プログラム	14	while (n>=10) {	0.1053
5	学習者プログラム	14	sum = 1;	0.0263

プログラムの一部を誤りに変更して作成したもの 4 つについて，初期値の定義が誤っているデジタルルートを求めるプログラムでは抽象度が低い場合に誤り箇所が上位 5 位に現れたが，疑惑値は非常に小さく，プログラムの大きさによっては上位 5 位に現れない可能性がある．また，抽象化レベルに応じて誤り箇所が上位に入るケースと入らないケースが存在した．学習者が起こしうる編集途中のもの 4 つについて，二桁のときの配慮をしていない誤りを含めたすべてのプログラムで模範解答プログラムの文が上位になったが，模範解答プログラムをそのまま学習者に返すことはできない．

表 5　デジタルルートの誤りプログラム 2 の疑惑値の上位 5 位 (抽象化レベル高)

疑惑値のついた文の説明				
順位	プログラムの種類	行数	文の内容	疑惑値
1	模範解答プログラム	16	while (n > 0) {	0.3044
2	模範解答プログラム	17	sum += n % 10;	0.1740
3	模範解答プログラム	18	n /= 10;	0.1740
4	模範解答プログラム	14	while (n >= 10) {	0.1305
5	模範解答プログラム	20	n = sum;	0.1305

表 6　デジタルルートの誤りプログラム 2 の疑惑値の上位 5 位 (抽象化レベル低)

疑惑値のついた文の説明				
順位	プログラムの種類	行数	文の内容	疑惑値
1	模範解答プログラム	16	while (n > 0) {	0.2609
2	模範解答プログラム	14	while (n >= 10) {	0.1740
3	模範解答プログラム	17	sum += n % 10;	0.1740
4	模範解答プログラム	18	n /= 10;	0.1740
5	模範解答プログラム	20	n = sum;	0.1305

3.3　誤り箇所推定方法の提案

3.2 節で得られた結果から学習者が誤り箇所の推定を行う際の精度を上げるための方法を提案する．

3.3.1　疑惑値の重み付け

3.1.4 節で示した疑惑値の計算方法では繰返しの回数が間違っていたときに差分として出てくる回数多くなり疑惑値が高くなりやすいが，初期値が誤りのプログラムのような 1 度しか実行されない文が誤っていた際に疑惑値が小さくなり，推定された誤り箇所の上位に現れない．

学習者プログラムのリテラルの代入文に疑惑値が付いた場合，その疑惑値を定数にする．また，繰り返しが 1 回多く実行された場合に繰り返し本体の文も 1 回多く実行される．すなわち，繰り返し文とその本体の文の疑惑値が等しい場合，誤りである可能性が高いのは繰り返しの文である．よって，繰り返し文の疑惑値を定数倍する．ここでは学習者プログラムのリテラルの代入分の疑惑値を 1.0 と定め，繰り返し文の疑惑値を 1.5 倍する．

3.3.2　抽象化レベルの異なる疑惑値の合成

抽象化レベルに応じて誤り箇所が上位に入るケースと入らないケースが存在し，誤り箇所に応じて適切に抽象化レベルを選択し誤り箇所の推定を行うことは難しい．2 つの抽象化レベルの疑惑値を合成することでより精度の高い疑惑値を計算できると考えた．抽象レベルの高い実行履歴は繰り返しなどの条件が省かれ，本研究で焦点を当てている「ある結果を得るための必要な計算とその回数」は抽出できている．抽象レベルの高い実行履歴に重きを置き，疑惑値の合成を行う方法を提案する．

ここでは抽象度高と抽象度低の疑惑値の比率が 1.5:1 になるように

$$S = \frac{(\text{抽象度高の疑惑値} * 1.5 + \text{抽象度低の疑惑値})}{2.5}$$

で求める．

3.3.3 模範解答プログラムのコメントの提示

学習者に対して疑惑値を付与した学習者プログラムを提示することは可能であるが，模範解答プログラムは直接提示できない．模範解答プログラムに学習者に提示するためのコメントを書いておき，模範解答プログラムの代わりに提示する方法を提案する．

3.3.4 結果

3.3.1，3.3.2，3.3.3 節で提案した手法をデジタルルートの誤りプログラム 1，2 に適用し，疑惑値の上位 5 位までまとめたものを表 7，8 に示す．

表 7 提案した手法を適用した誤りを含むデジタルルートのプログラム 1 の疑惑値の上位 5 位

疑惑値のついた文の説明					
順位	プログラムの種類	行数	文の内容	疑惑値	コメント
1	学習者プログラム	15	while (n > 0 || sum > 9) {	0.6079	
2	学習者プログラム	14	sum = 1;	0.4000	
3	学習者プログラム	16	if (n == 0) {	0.3947	
4	模範解答プログラム	14	while (n >= 10) {	0.2632	合計が一桁になるまで繰り返す
5	模範解答プログラム	16	while (n > 0) {	0.2237	各桁を全て足すまで繰り返す

表 8 提案した手法を適用した誤りを含むデジタルルートのプログラム 2 の疑惑値の上位 5 位

疑惑値のついた文の説明					
順位	プログラムの種類	行数	文の内容	疑惑値	コメント
1	模範解答プログラム	16	while (n > 0) {	0.4305	各桁を全て足すまで繰り返す
2	模範解答プログラム	14	while (n >= 10) {	0.2219	合計が一桁になるまで繰り返す
3	模範解答プログラム	17	sum += n % 10;	0.1740	1 の位の数字を取得して合計に加える
4	模範解答プログラム	18	n /= 10;	0.1740	1 の位を取り除く
5	模範解答プログラム	20	n = sum;	0.1305	合計を次の計算対象にする

提案した手法を適用した結果について，プログラムの一部を誤りに変更して作成したもの 4 つのうち，疑惑値の上位 5 位について初期値の定義が誤っているデジタルルートを求めるプログラムと最後の桁を足す計算が間違っているプログラムでは 2 位に初期値の誤りが現れた．繰り返し条件が&&になっているプログラムは繰り返しが一度も実行されなかったので，疑惑値の上位 5 位には模範解答プログラムのみ現れた．繰返し条件の範囲が異なるプログラムは繰り返しの実行回数が少なかったが，疑惑値の上位 5 位には現れず，6 位に現れた．学習者が起こしうる編集途中のもの 4 つについては，繰返し回数が不足しており疑惑値の上位 5 位には模範解答プログラムが多く現れた．

4 評価・考察

4.1 提案方法のパラメータの考察

繰返しの疑惑値の倍率を 1.1 倍にする，抽象度高と抽象度低の疑惑値の比率を 2:1 で合成するなど 3.3 節で提案したパラメータ以外を適用したが，疑惑値の数値が増減するのみで順位が大きく変動することはなかった．

テストケースにより繰返し回数などは変わるので，有効なテストケースについて今後検討が必要である．

4.2 適用例の評価

3 章で提案した手法を累乗を求めるプログラムと誤りを含むプログラムに適用し，疑惑値の上位 5 位に誤り箇所が現れるか調査する．繰り返し条件がカウントアップである $x^n = 2^5$ の累乗を求めるプログラムの模範解答プログラムと誤りを含むプログラムの疑惑値上位 5 位をまとめたものを表 9 に示す．誤りを含むプログラムは繰り返し条件をカウントダウンで制御しているかつその条件が誤っているもの (18 行目:$while(0 <= c)\{$) である．

表 9 誤りを含む累乗プログラム 1 の疑惑値の上位 5 位

疑惑値のついた文の説明					
順位	プログラムの種類	行数	文の内容	疑惑値	コメント
1	模範解答プログラム	22	i++;	0.3571	繰り返し回数の変更
2	学習者プログラム	20	c−;	0.3495	
3	学習者プログラム	18	while (0 <= c) {	0.2258	
4	模範解答プログラム	20	while (i <= n) {	0.1385	指数の回数繰り返す
5	模範解答プログラム	18	n = 5;	0.0428	指数入力

提案した手法を適用した結果より，プログラムの誤り箇所が疑惑値の上位 5 位に含まれている．繰り返し文の制御がカウントアップかカウントダウンのどちらかにより，本来誤りでない箇所に疑惑値がつくが，本来誤りである箇所との疑惑値の値の差は大きくないので誤り箇所の特定に大きく影響することはないと推測できる．

5 おわりに

本研究では，模範解答プログラムと誤りプログラムの実行履歴の比較による誤り箇所推定方法を考察した．必要な演算やその回数に焦点を当て，実行履歴を抽象化することで，プログラムの記述方法に左右されずに誤り箇所を推定できる可能性を示した．今後の課題として，疑惑値計算の精度の向上や誤り箇所推定ツールの実装，実データによる有用性の評価が挙げられる．

参考文献

[1] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, Franz Wotawal : A Survey on Software Fault Localization, IEEE Transactions on Software Engineering (Volume: 42, Issue: 8), pp. 707-740, 2016.

[2] Zheng Li, Xiaotang Zhou, Yonghao Wu, Yong Liu : Xiang Chen : Faulty Statement category Frequency-based Fault Localization (FSFFL), 2021 16th International Conference on Computer Science & Education (ICCSE), pp. 969-974, 2021.

[3] 田中裕人, 戸田順也 : C 言語学習者向けの実行履歴取得およびポインタ更新可能なプラットフォームの提案, 南山大学理工学部 2020 年度卒業論文 (2021).