プログラミング学習のための誤り範囲特定ツールの拡張方法の提案
―セグメント不足の対応と誤り特定時間削減に着目して―

2020SE024 近藤玲　2020SE085 吉岡永遠
指導教員：蜂巣吉成

1　はじめに

プログラミング演習において学習者は，与えられた演習問題に対するプログラムを作成し，期待した実行結果が得られるか自身で確認する．期待した実行結果が得られなかった場合，自力で誤りを特定したり，教員や TA による支援を受けるといった解決方法が存在するが，知識や経験の少ない学習者にとって，プログラムの誤り範囲の特定は難しい場合がある．また，教員や TA による対応ができる人数には限りがある．

この問題を解決するために，澤田ら [4] は，学習者が解答したプログラム (以下，修正対象プログラム) をセグメントと呼ぶコード片に分割し，修正対象プログラムのセグメント (以下，修正対象セグメント) を模範解答プログラムのセグメント (以下，模範解答セグメント) で置き換えたプログラム (以下，修正候補プログラム) を生成，実行し，誤り範囲を特定する方法を提案している．セグメントとは，分岐のない連続した文の集まりであり，制御文を基にプログラムをセグメントに分割する．しかし，学習者の記述不足で本来あるべきセグメントがない場合 (以下，セグメント不足)，誤り範囲の特定ができなかった．誤りが存在するセグメント (以下，誤りセグメント) が増加すると，修正対象セグメントと模範解答セグメントの置換数も増加するので誤り特定時間が長くなる．

本研究では，澤田らの方法を拡張し，記述されていないセグメントに対して文を挿入して誤り範囲を特定する方法を提案する．文を挿入した場合の誤り特定時間を分析し，実用的な時間で特定可能である誤りセグメントの組合せを確認する．セグメント不足の誤りを特定するツールを作成し，提案方法の有効性を確認した．また，効率的な特定を実現するために，澤田らのツールで誤り特定に時間のかかっている処理を分析し，それを削減する．

2　関連研究

2.1　SBFL と自動バグ修正手法の性能調査

Spectrum-Based Fault Localization(SBFL) では，テストケースによる実行経路の情報を用いて誤り範囲を推定する手法である [1]．本研究では，誤りのないプログラムが模範解答として存在するプログラミングの演習問題に対して，学習者の意図に近い誤り範囲の特定方法を提案する．

松尾らは，深層学習を用いて構文エラーの修正ができる DeepFix[2] を，教育支援の観点から調査を行っている [3]．本研究では，コンパイルが通るが正しい実行結果にならないものに対して，誤り範囲の特定方法を提案する．

2.2　澤田らの誤り範囲特定ツール

澤田らの研究では，プログラミング演習におけるセグメントを用いたソースコード誤り範囲特定方法を提案している [4]．修正対象プログラムをセグメントの定義に従い分割し，模範解答セグメントと差分を取る．ソースコード 1 の累乗を計算する演習問題の模範解答プログラムは「`int v1; double v2; v2=1;`」「`if (0<=p2)`」「`for (v1=0; v1<p2; v1=v1+1)`」「`v2=v2*p1;`」「`else`」「`for (v1=0; p2<v1; v1=v1-1)`」「`v2=v2/p1;`」「`return v2;`」のように 8 個のセグメントに分割される．修正対象セグメントを差分が小さい模範解答セグメントで置き換えた修正候補プログラムを生成，実行し，main 関数に用意した複数のテストケースを修正候補プログラムがパスしたならばそのセグメントを誤り範囲として特定する．修正候補プログラムには，コンパイルエラー，タイムアウト，失敗，成功の 4 つの状態が存在する．コンパイルエラーはコンパイルが通らなかったもの，タイムアウトは無限ループなどによって，プログラムの実行に 1 秒以上掛かるもの，失敗はコンパイルは通るが実行結果が正しくないもの，成功はテストケースをすべてパスしたものである．しかし，本来あるべきセグメントがないセグメント不足の場合，セグメント同士の比較ができないので誤り範囲の特定ができない．

セグメント不足について例を用いて説明する．ソースコード 2 は，2 箇所の誤りを含んだソースコード 1 の一部である．

ソースコード 1　累乗を計算するプログラム

```
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 2 誤りを含んだ累乗プログラム

```
10      } else {
11        for (v1 = 0; p2 < v1; v1 = v1 + 1) {
12          v2 = v2 / p1;
13        }
14      }
15    }
```

ソースコード 2 の 11 行目における for 文の誤りはセグメントの置き換えによって修正可能だが，14 行目の後に不足している return 文については，セグメントが記述されていないので追加することはできず，誤り範囲の特定ができない．本研究では，このような学習者の記述不足で特定できなかった誤り範囲を特定可能とする方法を提案する．

3 セグメント不足の誤り範囲特定方法

本研究における修正対象プログラムの前提条件は，澤田らと同じく「コンパイル可能だが，正しい実行結果ではない」「関数を作成する問題で，関数名と仮引数は与えられる」を満たし，main 関数に対象の関数をテストするための呼び出しが記述されているものとする．セグメント分割に関しても澤田らと同様の定義に従って分割していく．また，本研究で対象となりうる誤りの種類を誤り 2 箇所までとして考えると，次の 9 つに分類される．

- 澤田ら
  1. 誤りセグメント確定が 1 箇所
  2. 誤りセグメント不明が 1 箇所
  3. 誤りセグメント確定が 2 箇所
  4. 誤りセグメント確定が 1 箇所かつ不明が 1 箇所
  5. 誤りセグメント不明が 2 箇所
- 本研究
  6. セグメント不足の誤りが 1 箇所
  7. 誤りセグメント確定が 1 箇所かつセグメント不足の誤りが 1 箇所
  8. 誤りセグメント不明が 1 箇所かつセグメント不足の誤りが 1 箇所
  9. セグメント不足の誤りが 2 箇所

ソースコード 3 誤りセグメント確定の例

```
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 3 の 12 行目のセグメント「v2 = v2 + p1;」は，模範解答セグメントに同一のものが存在しないので差分がゼロでなくなり，誤りセグメントの位置と断定できる．これを「誤りセグメント確定」と表現する．

ソースコード 4 誤りセグメント不明の例

```
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 4 の 12 行目のセグメント「v2 = v2 * p1;」は，「v2 = v2 / p1;」となれば正しい実行結果が得られるが，模範解答セグメントに同一のものが存在するので差分がゼロとなり，誤りセグメントの位置と断定できない．これを「誤りセグメント不明」と表現する．

3.1 セグメント不足の分析

セグメント不足がどのようなものなのか分析する．プログラムにおける主な抜けは，変数宣言抜け，初期化抜け，return 文抜け，if，while，for 文などの制御文自体の抜け，複合文中の 1 文抜け，制御文以外の独立した文の抜け (前述したもの以外) などが挙げられる．

変数宣言の抜けは，コンパイルエラーとなり，前提条件のコンパイルが通るものを満たさないので，本研究では対象外とする．初期化は，その上で変数宣言を行なっている場合，変数宣言と初期化を合わせて 1 つのセグメントとするので，セグメント不足の誤りではなく，セグメント自体の誤りとなる．ただし，プログラムの途中などでその上で変数宣言等を行なっていない場合，セグメント不足の誤りとなる可能性がある．return 文抜けは，コンパイラによってコンパイルエラーとなる場合があるが，本研究では対象とする．制御文は，制御文の予約語と制御式を囲む丸括弧の部分 (以下，頭部) と，実行される文が書かれている波括弧の部分 (以下，本体) に分けられる．制御文はセグメント分割により，頭部と本体の 2 つに分割される．そのため，制御文の抜けは連続した 2 箇所の抜けとなり，組合せが多くなるので本研究では対象外とする．複合文は 2 文以上ある場合もまとめて 1 つのセグメントとするので，複合文中の 1 文が抜けている場合，セグメント自体の誤りとなる．

これらを考慮すると，本研究におけるセグメント不足は，主に一部の初期化抜け，return 文抜け，制御文以外の独立した文の抜けが該当する．

3.2 セグメント不足の修正方法

本研究では，空白のセグメント (以下，空セグメント) を用いたセグメント不足の誤り範囲特定方法を提案する．すべての修正対象セグメントの間に空セグメントを挿入し，それを用いて，セグメント不足の誤り範囲特定を行う．修正対象セグメントの間に挿入された空セグメントと模範解答セグメントを順に置換する．置換したセグメントと修正対象セグメントで修正候補プログラムを作成し，実行する．実行したプログラムがテストケースをパスした場合，置換したセグメントがセグメント不足の誤りとして特定される．このような特定方法にすることで拡張前ツールと同様の特定方法で拡張を行える．

3.3 セグメント不足を含む誤り範囲特定に要する時間

誤りの種類からツールが誤り範囲を特定するのに要する時間 (以下，特定時間) を分析し，実用的な時間で実行可能かどうかなど，大まかな時間を見積もり，対象とする誤りを決定する．本研究では，本来あるべきセグメントがない場合の誤り範囲の特定を目的としているので，誤りの種類の 6 から 9 の誤りについて，その中でもが大きくなると考えられる「8: 誤りセグメント不明が 1 箇所かつセグメント不足の誤りが 1 箇所」「9: セグメント不足の誤りが 2 箇所」について分析する．特定時間が最大となる場合の修正候補プログラム数を計算し，プログラムの作成と実行にかかる時間と積をとる．模範解答セグメントの総数を m，修正対象セグメントの総数を n，セグメント不足の誤りに対して模範解答セグメントを挿入する箇所の数を $n' = n + 1$ とする．

以下，$m = 100$, $n = 10$, プログラムの作成と実行に 0.1 秒要するとして計算する．なお，値については澤田らの研究 [4] と同じものを使用する．

- 誤りセグメント不明が 1 箇所かつセグメント不足の誤りが 1 箇所
  1. 誤りセグメント不明が 1 箇所と仮定
     ${}_nC_1 \cdot m = n \cdot m = 1000$
  2. 誤りセグメント不明が 2 箇所と仮定
     ${}_nC_2 \cdot m \cdot m = \frac{n \cdot (n-1)}{2} \cdot m^2 = 450000$
  3. セグメント不足の誤りが 1 箇所と仮定
     ${}_{n'}C_1 \cdot m = (n+1) \cdot m = 1100$
  4. 誤りセグメント不明が 1 箇所かつセグメント不足の誤りが 1 箇所と仮定
     ${}_nC_1 \cdot m \cdot {}_{n'}C_1 \cdot m = n \cdot (n+1) \cdot m^2 = 1100000$

1 から 4 の式をすべて足した 1552100 が最大の修正候補プログラム数である．これを時間に換算すると，およそ 43 時間となる．実際は差分のしきい値を用いて，修正候補プログラムを削減するので，特定時間は見積りより短くなる．しきい値は，学習者の意図に近い模範解答セグメントと誤りセグメントを置換するために用いる．修正対象セグメントごとに削減される修正候補プログラムの数は異なるが，ここではすべて一定数削減されるとする．ただし，セグメント不足の誤りは学習者の意図によらないので，すべての模範解答セグメントで試すべきであり，しきい値による候補削減はできない．修正候補プログラムが 20 まで削減されたとすると，特定時間はおよそ 6 時間 30 分となる．

「9: セグメント不足の誤りが 2 箇所」存在する場合は，上記の計算量に ${}_{n'}C_2 \cdot m \cdot m = 550000$(セグメント不足の誤りが 2 箇所と仮定) を加えた値となり，時間にしておよそ 15 時間増加する．しきい値を考慮すると，上記の例の場合，特定時間はおよそ 21 時間 30 分となる．

以上の結果より，誤りの種類の 6 から 8 の誤りに関して，実用的な時間での特定が可能と考えた．

3.4 セグメント不足を含む誤り範囲特定の順序

本研究のツールの誤り特定の流れは次のようにする．

- 誤りセグメント確定が 2 箇所
  1. その 2 箇所を置換
  2. 失敗したら，誤り特定不可
- 誤りセグメント確定が 1 箇所
  1. その 1 箇所を置換
  2. 失敗したら，誤りセグメント確定が 1 箇所かつ不明が 1 箇所として置換
  3. 失敗したら，誤りセグメント確定が 1 箇所かつセグメント不足の誤りが 1 箇所として置換
  4. 失敗したら，誤り特定不可
- 誤りセグメント確定が 0 箇所
  1. 誤りセグメント不明が 1 箇所として置換
  2. 失敗したら，誤りセグメント不明が 2 箇所として置換
  3. 失敗したら，セグメント不足の誤りが 1 箇所として置換
  4. 失敗したら，誤りセグメント不明が 1 箇所かつセグメント不足の誤りが 1 箇所として置換
  5. 失敗したら，誤り特定不可

本研究のセグメント不足の誤りを含む誤り範囲特定の順序は，澤田らの特定順序にセグメント不足を含む誤りを追加したものである．順序の最適化については 6 章で考察する．

4 誤り特定時間削減方法

澤田らのツールの誤り特定時間削減方法を提案する．

4.1 効率化すべき箇所の特定

効率化するためにどこに多くの時間を掛けているのかを特定する必要がある．2 個の誤り範囲がある 10 個の修正対象プログラムを実行し，コンパイル回数，コンパイルエラー，タイムアウト，失敗となるプログラム数と特定時間を調査したところ，タイムアウトが多いと特定時間がかかることが分かった．タイムアウトになった修正候補プログラムの特徴として，カウンタ変数を用いた for 文や while 文で無限ループになることが分かった．

4.2 誤り特定時間削減の実験

タイムアウトの修正候補プログラムをコンパイルしないことによって，実際に特定時間が短くなるのか調べた．属性付き字句系列に基づく書換え処理系である TEBA を用いて無限ループとなるプログラムのパターンを記述した [5]．澤田らのツールと本研究のツールで，特定時間を測定した結果を表 1 に示す．ツールは MacBook Pro(M1 pro 10 コア，16GB メモリ，macOS 14.2.1) で実行し，time コマンドで特定時間を 3 回計測して平均した．差分のしきい値は 10，タイムアウト 1 秒とした．ファイル名の最後の文字は，誤りの種類を表す．たとえば，forA と whileA で

は，誤り範囲が同じ for 文と while 文を書き換えた修正対象プログラムとなっている．また，ファイル名の最後の文字が A または B は誤り範囲を比較的見つけやすい修正対象プログラム，C または D は誤り範囲を比較的見つけにくい修正対象プログラムである．

表 1 より，ファイル名の forA，B と whileA，B の修正対象プログラムでは，本研究のツールの方が特定時間が長かった．また，forC，D と whileC，D の修正対象プログラムでは，本研究のツールの方が特定時間が短かい結果となった．

表 1 澤田らのツールと本研究のツールの特定時間の比較

	澤田らのツール			本研究のツール		
FN	CN(回)	TO(個)	ET(秒)	CN(回)	TO(個)	ET(秒)
forA	115	13	40.356	97	1	52.799
forB	28	0	14.221	28	0	18.518
forC	1758	420	724.511	931	7	477.536
forD	5769	2991	3503.138	92	1	929.418
whileA	202	31	79.829	187	18	108.913
whileB	1	0	10.225	1	0	10.892
whileC	2067	1033	1323.177	1000	275	874.762
whileD	1319	552	769.848	608	45	460.298

FN: ファイル名，CN: コンパイル回数，TO: タイムアウト，ET: 特定時間

4.3 澤田らのツールより誤り特定時間が増加するもの

本ツールではタイムアウトになりうるかどうかの判定をすべての修正候補プログラムにしている．したがって，4.2 節の誤り範囲を比較的見つけやすいプログラムのようなタイムアウトになりうると判断した数が少ない例では本ツールの特定時間の方がわずかに長くなる．

5 評価

作成したツールを評価するため，本研究におけるセグメント不足の定義を満たすプログラムを作成して実験を行った．該当する抜けは主に，1. return 文抜け，2. 一部の初期化抜け，3. 制御文以外の独立した文の抜けである．1，2，3 は該当するケースの番号 (以下，ケース) である．ケース 1 は，ソースコード 2 を用いる．ケース 2 は，ソースコード 1 の for 文を while 文に変換し，その while 文の初期化が抜けているものでテストする．ケース 3 は，文字列コピーのプログラムで，最後にナル文字の代入が不足しているものでテストする．なお，誤り特定時間の早いものと，遅いものとの比較や，3.3 節の見積もり時間との比較をするために「誤りセグメント確定が 1 箇所かつセグメント不足の誤りが 1 箇所 (表 2 では，確定と不足)」「誤りセグメント不明が 1 箇所かつセグメント不足の誤りが 1 箇所 (表 2 では，不明と不足)」の場合でテストする．実行環境は 4.2 節と同じである．見積もり時間と誤り特定時間を比較した結果を表 2 に示す．

ほとんどのテストケースにおいて見積もり時間内に誤り範囲特定ができた．見積もり時間との差は，しきい値による候補削減にばらつきがあることなどが理由である．

表 2 見積もり時間と誤り特定時間の比較

誤りの種類	ケース	m(個)	n(個)	見積もり時間	特定時間
不明と不足	1	80	7	2 時間 30 分	1 時間 11 分 50 秒
	2	80	9	4 時間 30 分	2 時間 56 分 42 秒
	3	30	7	1 時間	24 分 13 秒
確定と不足	1	80	7	25 分	25 分 25 秒
	2	80	9	32 分	23 分 2 秒
	3	30	7	12 分	4 分 36 秒

m: 模範解答セグメント総数，n: 修正対象セグメント総数

6 考察

3.4 節の特定順序は最適とは言えない．3.3 節から特定に時間のかかる誤りは，誤りセグメント不明が 2 箇所，誤りセグメント不明が 1 箇所とセグメント不足の誤りが 1 箇所である．これらの誤りのうちどちらを先に特定するかによって特定時間に違いが出る．誤りの傾向やコンパイラの警告などでどちらを優先するか決めると良い．例えば，未初期化の変数の使用や return 文がない実行経路について警告を表示するコンパイラがあるので，これらに該当した場合はセグメント不足を含む誤り特定を優先して行う．

7 おわりに

本研究では，学習者の記述不足で本来あるべきセグメントがない場合，誤り範囲の特定ができないとった問題を解決することを目的として，ツールの拡張方法を提案した．今後の課題として，誤り傾向や，コンパイル時の警告を利用したセグメント不足の誤りの特定順序の検討などが挙げられる．

参考文献

[1] W. E. Wong, R. Gao, Y. Li, R. Abreu and F. Wotawa: A Survey on Software Fault Localization, IEEE Transactions on Software Engineering (TSE), Vol.42, No.8, pp.707-740 (2016).

[2] Gupta, R., Pal, S., Kanade, A., and Shevade, S.: DeepFix: Fixing Common C Language Errors by Deep Learning, in Proceedings of the 31st AAAI Conference on Artificial Intelligence, pp. 1345-1351 (2017).

[3] 松尾 春紀, 池田 翔, 亀井 靖高, 佐藤 亮介, 島田 敬士, 鵜林 尚靖：教育支援の適用に向けた自動バグ修正手法の性能調査, Vol. 38, No. 4, pp. 4_16-4_22 (2021).

[4] 澤田 侑希, 梅田 祐一郎, 蜂巣 吉成, 吉田 敦, 桑原 寛明：プログラミング演習におけるセグメントを用いたソースコードの誤り範囲特定方法の提案コンピュータソフトウェア, Vol. 40, No. 4, pp. 4_29-4_36 (2023).

[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換え支援環境, 情報処理学会論文誌, Vol. 53, No. 7, pp. 1832-1849 (2012).