実行経路に基づく適用可能なライブラリ関数の特定支援手法の提案
-文字列処理を対象として-

2020SE059 下平健太　2020SE069 田中智規
指導教員：吉田敦

1　はじめに

標準ライブラリには頻繁に使われる機能があらかじめ関数として定義されており，これらを用いることで簡潔にプログラムを記述できる．また，ライブラリ関数を再利用することで，開発効率の向上だけでなく，信頼性の向上やリスクの軽減が期待される．標準ライブラリであれば，共通認識があるので，関数名のみで機能を理解できる．

プログラミング学習者がプログラムを書く際，自分で機能の実現が可能な場合，標準ライブラリ関数が利用可能なことに気づかないことがある．そのようなプログラムは可読性が低下し，理解に時間がかかる．また，標準ライブラリ関数に相当する処理が他の処理と融合している場合，プログラムが複雑化し，保守性が下がる．

学習者が標準ライブラリ関数を用いずプログラムを書くといった問題は，文字列処理のライブラリでは特に発生しやすい．printf などの入出力関数についての標準ライブラリ関数では学習者はその標準ライブラリ関数を用いなければ望む機能を実現できない．一方で文字列処理は，基本的な機能の実現をプログラミング学習の中で学び，標準ライブラリ関数を使わずに書ける．文字列だけでなく，配列やポインタの課題としても文字列処理を扱う．よって，学習者は文字列操作に関する標準ライブラリ関数を使用する機会が少なく，どの関数が使えるか気付けない．そこで，学習者のプログラム内で適用可能な標準ライブラリ関数特定支援手法を提案する．本研究では C 言語の文字列の基本操作を対象とする．関数ごとに実行経路を比較することで，表現の違いを吸収しつつ，適用可能性を表す確信度を算出して提示する．

技術的課題として以下のことがあげられる．

(1) 記述中のプログラムの中に標準ライブラリ関数の処理が融合して存在すると，単純な比較では特定できない．

(2) 学習者が標準ライブラリと同じ機能を書いていても，その記述が標準ライブラリと一致するとは限らない．

(3) 標準ライブラリ関数の基本的な機能のみが学習者のプログラムに含まれると，全体の類似具合が低く，高くあるべき確信度も低下する．

ソースコード 1　プログラムの例

```
1  while (str1[i] != '\0') {
2      i++;
3  }
4  while (str2[j] != '\0') {
5      if(str2[j] >= 'a' && str2[j] <= 'z'){
6          str1[i] = str2[j] - 'a' + 'A';
7      } else {
8          str1[i] = str2[j];
9      }
10     i++;
11     j++;
12 }
```

(1) に対する例をソースコード 1 に示す．全体は strcat の処理であるが，5, 6 行目に toupper の処理が混在している．このように 1 つの機能内に他の処理を実現する記述が融合して存在すると，単純な比較では特定できない．

(2) の例として for や while などの繰り返しの違いといった異なる記述方法が存在するため，表記の違いを吸収する必要がある．文字列処理のライブラリ関数は各機能が単純で，基本的なアルゴリズムが変わることは少ない．

(3) に対して多くの関数では，その機能を特徴付ける文が存在する．2 つの文字列を連結する str1[i] = str2[j]; という文が strcat 関数を特徴付ける文である．i++; のような様々な関数に出現する文の有無よりも機能を特徴付ける文の有無に対して確信度はより依存すべきである．関数を特徴付ける文を考慮した比較方法が必要である．

課題解決のために実行経路を用いた比較を行う．実行経路とはプログラムが実行される際に，どの命令がどの順序で実行されるかを示すものである．本研究での実行経路は条件式とその真偽，式文によって構成されているものとする．ソースコード 1 のような複数の処理が 1 つの繰り返しにまとまっている場合，標準ライブラリ関数に相当する処理が if で分岐しているなら比較に用いる標準ライブラリ関数の実行経路に類似した実行経路が存在している．分岐しないなら，学習者のプログラムから得た実行経路には標準ライブラリ関数の実行経路を包含する実行経路が存在しているので，適用可能な標準ライブラリ関数の機能を含むかどうかを判定できる．また，異なる記述であっても処理内容が同じであれば制御に関わる違いは実行経路ごとに比較することで吸収できる．他の表現の違いについては差分をとる際に実行経路の要素同士の等価性を判定することで吸収する．課題 (3) については，実行経路同士の比較の際に，標準ライブラリ関数側の機能を特徴付ける文に重み付けをした確信度を算出する．本研究では前提として学習者と標準ライブラリでは同じ変数名を使用するものとする．

2　関連研究

黒木らの研究 [1] では，類似するソースコードの存在を命令，基本ブロック，制御構造，条件式の 4 つの要素ごとの類似度から算出している．制御フローグラフと高水準中間表現 HIR を用いることでソースコードを抽象化すると

共に，データ依存情報も用いることで同一アルゴリズムを実装した複数のソースコードの同一性を判定している．4つの要素の類似度から同一性を判定するため記述法の違うソースコードでも判定可能である．しかし，目的の処理以外の処理が混在する場合では類似度が低くなり，特定のアルゴリズムを含んだソースコードを探すことは難しい．本研究では実行経路に着目することで，処理が混在する場合でも標準ライブラリを適用可能な箇所の特定をする．また，重みを付けることでより意味的に類似するものを発見し標準ライブラリの適用可能性の度合いを計算している．

3　適用可能なライブラリ関数の特定支援手法

3.1　システムの構成

本研究の学習者のプログラムの入力から関数ごとの確信度とその実行経路を提示するまでの全体の処理の流れを図1に示す．

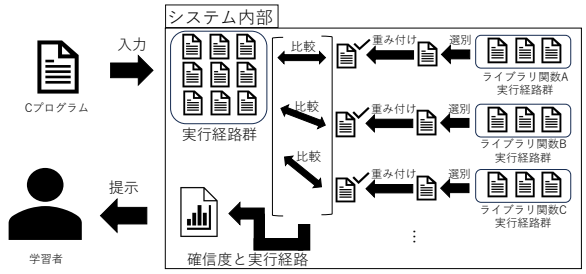

図1　関数ごとの確信度とその実行経路の提示までの流れ

学習者が記述したプログラムの制御フローグラフ (以下，CFG) から実行経路を求め，実行経路ごとに標準ライブラリと比較する．学習者が入力したソースコードから取得するすべての実行経路と比較対象の標準ライブラリは実行経路を選別し重み付けしたものとを比較する．

3.2　ソースコードから実行経路の取得について

学習者が入力したソースコードを CFG に変換し，分岐があった場合は各分岐ごとに経路を求め，各分岐ごとの経路をすべて求める．経路には各分岐で真偽のどちらの経路を進んだ場合かを表す情報を含む．ただし，繰り返しが存在すると実行経路の数は無限になるので，繰り返し回数に制限を設ける必要がある．実行経路の比較にあたって，繰り返しの中の後の文は，そのあとに前の方の文が実行される関係も反映させる必要があるので，繰り返しを複数回通る実行経路が必要である．一方，繰り返しの回数が増えるとその繰り返し数が確信度の値に強く影響を与えるので，必要最小限の2回とする．ソースコード3はソースコード2から得られる実行経路の1つである．各行は辿ったノードを表し，(#true) や (#false) がある行は条件式と真偽のどちらの経路に進んだかを表す．

ソースコード 2　strlen 関数を実現したソースコード

```
1 while (str1[i] != '\0') {
2   i++;
3 }
```

ソースコード 3　ソースコード 2 の実行経路

```
1   str1[i] != '\0' (#true)
2   i++;
3   str1[i] != '\0' (#true)
4   i++;
5   str1[i] != '\0' (#false)
```

3.3　比較に用いる標準ライブラリ

前提条件として，システムを利用する対象は学習者であり，記述されるプログラムは一般的に定義される標準ライブラリ関数と同一の記述であると限らない．ソースコード4は NetBSD[3] で定義された strcat 関数であり，学習者が素直に書くような記述ではない．比較に用いる標準ライブラリ関数は学習者が記述するものと合わせる必要があるので，一般的な標準ライブラリ関数の機能と同等になるよう独自に定義したものを使用する．定義するものは基本を学んだ学習者が素直に書いたものになるようにする．また，繰り返しの違いや条件文の違いは実行経路を用いることで吸収し，インクリメント，デクリメントの違いと条件式の違いの一部は実行経路の要素の比較時に行う要素同士の等価性の判定処理において吸収する．本研究では前提条件として，学習者のプログラムと標準ライブラリの識別子は一致しているものとする。.

ソースコード 4　NetBSD で定義された strcat

```
1 char *t = s;
2 _DIAGASSERT(t != NULL);
3 for (; *t; ++t)
4         ;
5 while ((*t++ = *append++) != '\0')
6         ;
7 return (s);
```

3.4　標準ライブラリ関数の比較に用いる実行経路

標準ライブラリ関数の実行経路にはソースコード5のように条件がすべて false のみの実行経路や，偽の経路を進むことで，標準ライブラリ関数として機能するための処理が行われない実行経路が存在する．学習者の入力ソースコードの実行経路にソースコード5と同じ実行経路が存在するからといって学習者に対し標準ライブラリ関数を適用可能だと考えることは不適切である．そこで，学習者の入力したソースコードから取得する実行経路に対して比較する標準ライブラリ関数の実行経路は選別する．選別する標準ライブラリ関数の実行経路は，最長のものとし，これを比較に用いる標準ライブラリ関数の実行経路とする．

ソースコード 5 strcat 関数の条件がすべて偽の実行経路

```
1  str1[i] != '\0' (#false)
2  str2[j] != '\0' (#false)
```

3.5 実行経路の比較

学習者の入力ソースコードの実行経路と各標準ライブラリ関数の適切な実行経路それぞれとの差分を求める．一般的に差分が少ないほど標準ライブラリ関数が適用可能である可能性が高くなる．得られる差分には以下の 3 つのノードが存在する．

(1) 共通のノード

(2) 学習者の実行経路にしかないノード

(3) 標準ライブラリ関数の実行経路にしかないノード

まず，確信度の概念を説明するために重み付けなしで説明し，そのあと重み付きに拡張する．確信度 Con は次の式にて求める．

$$Con = \frac{com}{com + stu + lib}$$

com : 共通するノードの数
stu : 学習者の実行経路にしかないノードの数
lib : 標準ライブラリ関数の実行経路にしかないノードの数

確信度の最大値は，完全に一致するときの値であり，1 となる．最小値は完全に一致しないときの値であり，0 である．

確信度の求め方をソースコード 1 を例に説明する．ソースコード 1 の 5 行目の if 文が false の場合ソースコード 6 の実行経路が存在する．

ソースコード 6 ソースコード 1 のとある実行経路

```
1  str1[i] != '\0' (#true)
2  i++;
3  str1[i] != '\0' (#true)
4  i++;
5  str1[i] != '\0' (#false)
6  str2[j] != '\0' (#true)
7  str2[j] >= 'a' && str2[j] <= 'z' (#false)
8  str1[i] = str2[j];
9  i++;
10 j++;
11 str2[j] != '\0' (#true)
12 str2[j] >= 'a' && str2[j] <= 'z' (#false)
13 str1[i] = str2[j];
14 i++;
15 j++;
16 str2[j] != '\0' (#false)
```

次に標準ライブラリ関数の strcat 関数を記述し実行経路を取得した比較に用いる実行経路がソースコード 7 である．

ソースコード 7 strcat 関数の比較に用いる実行経路

```
1  str1[i] != '\0' (#true)
2  i++;
3  str1[i] != '\0' (#true)
4  i++;
5  str1[i] != '\0' (#false)
6  str2[j] != '\0' (#true)
7  str1[i] = str2[j];
8  i++;
9  j++;
10 str2[j] != '\0' (#true)
11 str1[i] = str2[j];
12 i++;
13 j++;
14 str2[j] != '\0' (#false)
```

2 つの実行経路から得られる差分から，$com = 14, stu = 2, lib = 0$ が求まるので，確信度 Con は

$$Con = \frac{14}{14 + 2 + 0} = 0.875$$

となる．それぞれ確信度を求め，その中で最も値が大きいものを，ソースコード 1 に対する strcat 関数の確信度とする．

3.6 重み付き確信度

1 章で挙げた問題点 (3) を解決するには，入力されたプログラムから得られた経路を解析するにあたって，標準ライブラリ関数の処理を実現するための関数を特徴付ける文を考慮するよう拡張する必要がある．そこで，関数を特徴付ける文に重みをつけた重み付き確信度を定義する。重みは関数の文または条件式に 1 以上の値で付与する．重みが大きいほど，重要であることを意味する．差分から重み付き確信度 Con_w は次の式にて求める．

$$Con_w = \frac{\sum_{n=0}^{com} com_{w_n}}{\sum_{n=0}^{com} com_{w_n} + \sum_{n=0}^{stu} stu_{w_n} + \sum_{n=0}^{lib} lib_{w_n}}$$

com_w : 共通するノードの重み
stu_w : 学習者の実行経路にしかないノードの重み
lib_w : 標準ライブラリ関数の実行経路にしかないノードの重み

重み付き確信度も最大値は 1，最小値は 0 である．

4 実装

学習者に提示するシステムは PHP を用いた Web システムとして実装した．CFG の作成には TEBA[2] を用い，分岐の情報と共に文字列化して出力する．実行経路の比較には Algorithm::Diff[4] を用いた．

本研究では比較に用いる標準ライブラリ関数を独自に定義し実行経路の選別を行っている．重みは理想的には関数のソースコードに付与するべきだが，手動で実行経路の記述に追加している．

学習者のソースコードの保存や取得した実行経路すべてをある標準ライブラリ関数の実行経路との比較，比較して得られた確信度の最大値，確信度が最大の実行経路を出力を PHP にて行っている．確信度は百分率にて提示する．

5　評価

本研究の目的は学習者のプログラム内で，適用可能な標準ライブラリ関数の特定支援をすることである．

ライブラリ関数を 30 個用意し，そのライブラリ関数に対して，4 種類のテストケースを用意した．それぞれのテストケースに対して，5 個ずつプログラムを作成し，作成した 20 個のプログラムに対し提案手法を適用した。.

1 いずれかのライブラリ関数と完全一致する．

2 いずれかのライブラリ関数を包含する記述を持つ．

3 いずれかのライブラリ関数と一部が一致する．

4 いずれのライブラリ関数とは一致しない．

各種類ごとに期待する確信度の値は 1 は確信度が 1，2 は学習者の実行経路にしか存在しない記述が増えるほど確信度が低下することである．また，3 は標準ライブラリ関数の実行経路にしか存在しない記述が増えるほど確信度が低下，4 は確信度 0 が期待される出力である．

技術的課題が解決されているかを評価するために，同じ動作だが制御文等の書き方が異なるものを用意し作成したシステムに適用した．for 文 while 文の違いなどを含むプログラムに対して確信度が一致したので繰り返しの違いを吸収できることを確認した．3 のテストケースでは基本的な処理が混在しているプログラムを用意し実行したところ，基本的な処理が実現されている標準ライブラリ関数がそれぞれ高い確信度となることが確認された．これらのことから技術的課題が解決されていることが確認できた．

次に重み付けの有効性を評価する．学習者の実行経路に関数を特徴付ける文が含まれる場合，確信度は高くなり，含まれない場合は低くなることを確認した．例としてソースコード 1 を入力とし，重みの有無による確信度の変化が表 1 である．ソースコード 1 では strcat 関数と toupper 関数と strlen 関数が適用可能である．重み付け後，適用可能な関数の確信度が高くなり，他の関数の確信度が低くなることが確認できる．

表 1　確信度一覧

	strcat	toupper	strlen	strchr	strcpy	strcmp
重みなし	0.89	1	1	0.71	0.62	0.25
重みあり	0.90	1	1	0.45	0.42	0.17

6　考察

提案手法の問題点を検討する．本研究では候補内に標準ライブラリ関数にはない処理が多くなるほど確信度が下がる仕様になっている．候補内の処理が増えることにより確信度が下がることは妥当である．しかし，デバック用の記述であったり標準ライブラリ関数の処理を実現する際に使用される変数や配列に影響のない記述が存在する場合にも下がることは不適切である．そのため，影響のない記述は無視して比較する必要がある．

次に記述方法の違いに対して for 文や while 文の繰り返しの記述の違いや一部の表現の違いは吸収できた．しかしポインタでの記述など他の記述方法の違いをすべて吸収することはできていない．

また，比較に用いる標準ライブラリ関数の実行経路は手動で選別と重み付けをしている．これらの処理を自動化することが必要である．

さらに，変数名や配列名の違いに対して，本研究では前提条件で変数名，配列名は同じとしているが実際の学習者のソースコードは変数名，配列名が違うためそれぞれを対応付ける必要がある．対策としては，TEBA[2] には IdUnify といったモジュールがあり，2 つの構文木の内部識別記号を揃えることで構文木の要素を対応付けている．このモジュールを用いることにより，入力されたソースコードの変数名と標準ライブラリ関数の変数名の違いが吸収されると考える．

最後に本研究では各標準ライブラリ関数の確信度と最高確信度が出力される実行経路を特定している．しかし、編集を支援するためにソースコードの標準ライブラリ関数適用可能箇所を特定することが望ましい．対策としては，CFG から得られた実行経路の構文木情報を保持したままの比較により適用可能な箇所の特定ができると考えられる．

7　おわりに

本研究では，C 言語の文字列処理を対象に実行経路に基づく適用可能なライブラリ関数の特定支援手法を提案した．提案した方法を実現したツールを作成し，テストケースを用いて妥当性を確認した．今後の課題は，考察で述べた点の改善である．

参考文献

[1] 黒木さやか，上田高徳，平手勇宇，山名早人：プログラムコードの抽象化を利用した類似ソースコード検索システム，電子情報通信学会第 19 回データ工学ワークショップ B10-2

[2] 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).

[3] NetBSD ： 入 手 先<https://github.com/NetBSD/src/blob/trunk/common/lib/libc/string/strcat.c> (参照 2024 年 1 月 31 日)

[4] Algorithm::Diff ： 入 手 先<https://metacpan.org/pod/Algorithm::Diff> (参照 2024 年 1 月 31 日)