5ZH-09

オンライン・ジャッジ・システムにおける テストケース自動生成手法の検討

塩見　健太　　朝倉　宏一
大同大学大学院 情報学研究科 情報学専攻

１．はじめに

プログラミング教育が小中学校の学習指導要領に新たに導入されたことにより、プログラミング学習の需要が高まりつつある[1]。そして、授業や自宅でプログラミングを学習するためのシステムとして、プログラミング学習支援システムの活用が考えられる。プログラミング学習支援システムには、オンライン・ジャッジ・システムと呼ばれる学習者が提出したプログラムが正しいかどうかを自動的に判定するシステムが備わっている。

オンライン・ジャッジ・システムでは、プログラムへの入力とそれに対する正しい出力が記述されているテストケースを複数用意することで、プログラムの正誤を判定している。したがって、出題者・教授者は学習者の解答プログラムが出題意図を満足することを保証する複数の適切なテストケースを用意しなければならない。一般的に、この作業は大きな労力を伴う。

本稿では、オンライン・ジャッジ・システムにおいて、学習者の解答プログラムが出題意図を満たすかどうかを判定できるテストケースを自動生成する手法について検討する。出題者・教授者が用意した正解プログラムを分析し、プログラム中の実行フローと、その実行フローが実行される条件を抽出することで、テストケースを自動生成する方法について述べる。

２．提案手法

本稿では、出題者・教授者の用意した正解プログラムからテストケースを自動生成するために以下の手順を踏む。ここでは、プログラミング言語として C を使用する。また、プログラムに対する入力を処理する「値入力」と、プログラムが正しく動作しているかを確認するために重要な「if 文」を分析の対象とする。

A Method for Generating Test Cases in Online Judge Systems.
Kenta Shiomi and Koichi Asakura, Department of Informatics, Graduate School of Informatics, Daido University.

表１ ブロックごとに抽出する情報

ブロック	抽出する情報
BASIC	なし
IF	条件式、True 時の実行フロー、False 時の実行フロー
INPUT	入力変数名、型

１．正解プログラムから、すべての実行フローを抽出する
２．各実行フローが実行されるための条件を抽出する
３．各実行フローの条件が成り立つための入力値を作成する
４．作成された入力値に基づきテストケースを生成する

以下、実行フローを抽出するための手法について説明する。

＜2・1＞実行フロー抽出手法　LLVM[2]を使用して入力されたプログラムを解析し、抽象構文木(AST)を出力する。AST の各ノードを分析し、プログラムを対応するブロックに分類して実行フローを抽出する。今回は「if 文」と「値入力」を分析対象としているため、ブロックは if 文を示す「IF」、値入力を示す「INPUT」、それ以外を示す「BASIC」の三種類に分類し抽出する。

次に、分類した各ブロックを分析する。分析で抽出する情報は表 1 の通りである。今回分析対象としている IF ブロックの条件式に関しては、条件式を表す AST を解析することで抽出する。また、True 時、False 時の実行フローは、それぞれの AST を再帰的に解析して抽出する。

３．評価実験

提案手法の有効性を確認するため、図１に示す例題プログラムの実行フローと実行条件を抽出した。例題プログラムには if 文が 2 つ存在するため、三種類の実行フローが存在することがわかる。

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int i;
6      scanf("%d", &i);
7      if (0 <= i && i <= 9) {
8          if (i == 5) {
9              printf("i is 5.");
10         } else {
11             printf("i is 0 to 9.");
12         }
13     } else {
14         printf("i is not 0 to 9.");
15     }
16     return 0;
17 }
```

図１ 評価実験用プログラム

```
----- Route 1 -----
1. BASIC (DECL_STMT)
2. INPUT (CALL_EXPR)
3. IF (IF_STMT)
4. IF (IF_STMT)
5. BASIC (CALL_EXPR)
6. BASIC (RETURN_STMT)
History  : True -> True
Condition: [(0 <= i && i <= 9) && (i == 5)]
(i = 5)

----- Route 2 -----
1. BASIC (DECL_STMT)
2. INPUT (CALL_EXPR)
3. IF (IF_STMT)
4. IF (IF_STMT)
5. BASIC (CALL_EXPR)
6. BASIC (RETURN_STMT)
History  : True -> False
Condition: [(0 <= i && i <= 9) && !(i == 5)]
(i = 8)

----- Route 3 -----
1. BASIC (DECL_STMT)
2. INPUT (CALL_EXPR)
3. IF (IF_STMT)
4. BASIC (CALL_EXPR)
5. BASIC (RETURN_STMT)
History  : False
Condition: [!(0 <= i && i <= 9)]
(i = 51)
```

図２ 出力結果

実行結果を図２に示す。図２の「Route 1」から「Route 3」より、三種類の実行フローが抽出されていることがわかる。また、それぞれの実行フローでは実行されるブロックが、LLVM の AST で用いられている種類とともに示されている。

さらに、History の部分には、実行フローが実行される if ブロックの条件が示されている。例えば、Route 1 には 3, 4 番目のブロックに if ブロックが存在するが、それぞれの条件が True, True のときにこの実行フローが実行されることを示している。

また、Condition には、この実行フローが実行されるときの実際の条件式が抽出されており、この条件を満たす変数の値が「(i = 5)」として示されている。

図２の実行結果より、本手法で三種類の実行フローと、その実行フローが実行される条件が適切に抽出されていることがわかる。また、それぞれの実行フローが実行される変数の値も例示されており、これを用いることですべての実行フローに対するテストケースを生成可能であることがわかる。

４．おわりに

本稿ではテストケースを手動作成する際に多大な時間がかかってしまう問題に対して焦点を当て、テストケースの自動生成手法について検討した。評価実験により提案手法の有効性を確認でき、テストケースの自動生成が可能であることを示した。今後の課題として、以下の２つが挙げられる。

１つ目に、条件式の簡略化である。本手法では IF 文の条件式を and で繋げることによって抽出を行っている。しかし、中には絶対に実行されることのない矛盾が生じた条件式も存在し、この抽出手法では矛盾が生じているかを判断するのは難しい。条件式を簡略化することで、実行フローが実行される変数値を効率よく抽出することが可能である。

２つ目に、FOR 文を用いたプログラムや再帰呼び出しの解析の実装である。教科書に記載されるプログラムでは、「-1 が入力されるまでループする」や、「入力された値分繰り返す」といったプログラムが存在する。現状、それらのプログラムを解析することができないため、実装が必要である。

文　献

(1)文部科学省：学習指導領域「生きる力」, <URL: https://www.mext.go.jp/a_menu/shotou/new-cs/index.htm> (2021 年 12 月閲覧).
(2) LLVM Developer Group.：The LLVM Compiler Infrastructure, <URL: https://llvm.org/>.