

卒 業 論 文

プログラミング演習における模範解答を用いた テストケース評価基準の自動生成

Automatic generation of evaluation criteria for test cases
using example codes in programming exercises

指 導 教 員 中村 正樹 准教授

富山県立大学工学部 電子・情報工学科

学 籍 番 号 : 1515015

氏 名 尾崎 裕樹

提 出 年 月 2019 年 2 月

目次

第 1 章	はじめに	1
1.1	背景	1
1.2	目的	4
1.3	論文の構成	5
第 2 章	プログラミング演習におけるテストケース評価システム	6
2.1	テストケース評価システム	6
2.2	入力のデータ構造の定義	7
2.3	関数定義	8
2.4	評価基準の具体例	8
2.5	システムの評価	9
2.6	システムの課題	10
第 3 章	テストケース評価基準の自動生成	11
3.1	手法の概要	11
3.2	入力データの型に基づく評価基準	12
3.3	条件文に基づく評価基準	13
第 4 章	プログラミング課題への適用	21
4.1	評価基準を生成できた問題	21
4.2	評価基準を生成できなかった問題	25
4.3	繰り返し文を含むプログラムでの検証	27
4.4	配列を含むプログラムでの検証	27
4.5	関数を含むプログラムでの検証	28
第 5 章	おわりに	32
	謝 辞	33
	参 考 文 献	34

第1章 はじめに

本研究では，プログラミング教育において，教員が学生の作成したプログラムに対して適切な評価を行う方法を考え，その際に利用できるシステムを提案する。

1.1 背景

プログラミング演習においては，教科書のソースコードを書き写してプログラムを作成することから学習を始める学生もいる。教科書のコードが正確に書き写しているかの確認をするには，教科書のソースコードをと学生のソースコードを一字一句比較すればよい。教科書のソースコードを書き写すだけではプログラムを作成できるようなのではないので，プログラミング演習では提示された仕様（課題や問題）を満たすプログラムを作成することを学ぶ必要がある。仕様を見てプログラムを作成した場合，一般に仕様を満たすプログラムは，一つに限らないため教員と学生が同じ実装をするとは限らず，教員の用意する模範解答のソースコードと学生のソースコードを単純に比較しただけでは正確な評価はできない。

通常，プログラムが仕様を満たすかどうかはソフトウェアテストによって確かめられる。文献 [1] を参考にソフトウェアテストについて説明する。

- ソフトウェアテスト
 - ー プログラムにテストデータを与えて実行した結果からソフトウェアの開発中に紛れ込んだ誤りを検出する作業である。
 - ー ソフトウェアテストのテスト技法には，ホワイトボックステストとブラックボックステストがある。
- ホワイトボックステスト
 - ー ソフトウェアの内部仕様やプログラムコードに基づいてテストケースを作成し，テストを実施する。
- ブラックボックステスト
 - ー ソフトウェア内部の詳細を見ずに，プログラムに入力データを与え，実行結果だけを観測することで誤りを検出する。
- テストケース
 - ー ソフトウェアテストを行うための入力と出力の組のことである。

教員がテストケースを用意する場合のプログラミング演習では、学生は用意されたテストケースでプログラムのテストを行うことで、プログラムが仕様を満たしているかを確認することができる。しかし、学生がテストケースを設計することを演習の対象とする場合も考えられる。学生が設計したテストケースが適切であるか評価するには、ソースコードと同様に、教員のテストケースと学生のテストケースを一字一句の比較による評価では不十分である。例えば、整数を入力して値が正の場合と負の場合、0の場合でそれぞれ異なる文字を表示するかをテストする場合を考える。学生と教員がテストケースの入力を表 1.1 のように用意したとする。教員テストケースと学生のテ

表 1.1 学生と教員のテストケース

	学生	教員
テストケース 1	1	2
テストケース 2	0	0
テストケース 3	-4	-3

トケースを比較すると正の場合と負の場合の入力が異なっているため、一字一句の比較では学生のテストケースは誤りと判断される。しかし、学生のテストケースでも 3 つの処理を確認することができるため、実際には誤りではない。したがって、学生の設計したテストケースが適切であるかを評価するための方法を考える必要がある。本研究におけるテストケース評価方法の位置づけを図 1.1 に示す。

テストケースを評価する方法の例として、文献 [2] では、テスト駆動開発に基づきプログラミング演習を行うことによってコーディングだけでなく、ソフトウェアテストの学習が行われる。評価対象は学生のプログラムとテストケースで、教員の作成したプログラムとテストケースを用いて図 1.2 のようにトリプルチェックテストを行うことによって評価される。

テストケースチェックによって、仕様要求を満たしていないテストケースが検出され、学生のテストケースの間違いが早期に指摘される。なお、仕様要求とは、プログラムにおける関数あるいはサブルーチンの動作が規定されたものであり、戻り値や引数などが定められたものである。教員のプログラムとテストケースはこの仕様要求が満たされるとする。次のセルフチェックでは、テストの実行だけでなくコードカバレッジの測定が行われる。テストにパスしない場合は、学生のプログラムが正しく実装されていないことが分かり、テストにパスしたが、カバレッジが 100% でない場合は、テスト漏れが検出され学生に指摘される。最後の実装チェックでは、学生のプログラムが仕様要求の要求をすべて満たしていないことが検知され、次のテスト駆動開発サイクルを始めるよう学生に通知される。実装チェックをパスすることによって、学生が作成したプログラムとテストケースが仕様要求の要求をすべて満たしたことが確認される。

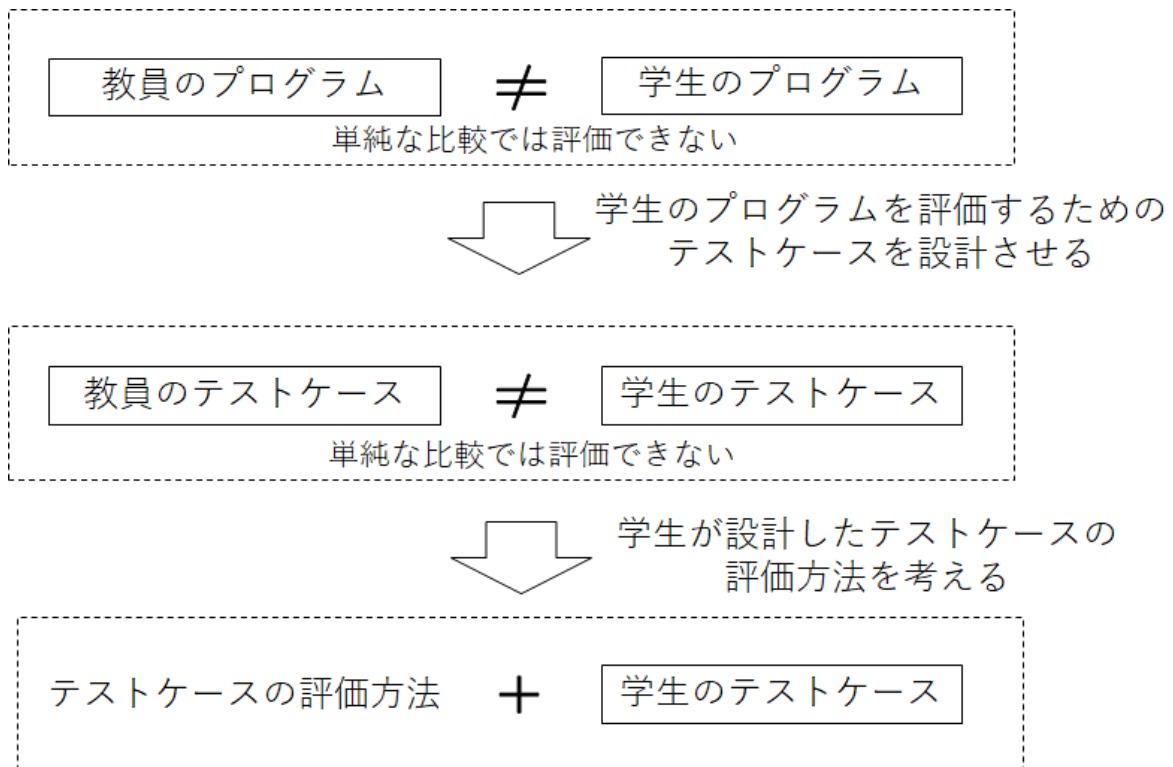


図 1.1 テストケース評価方法の位置づけ

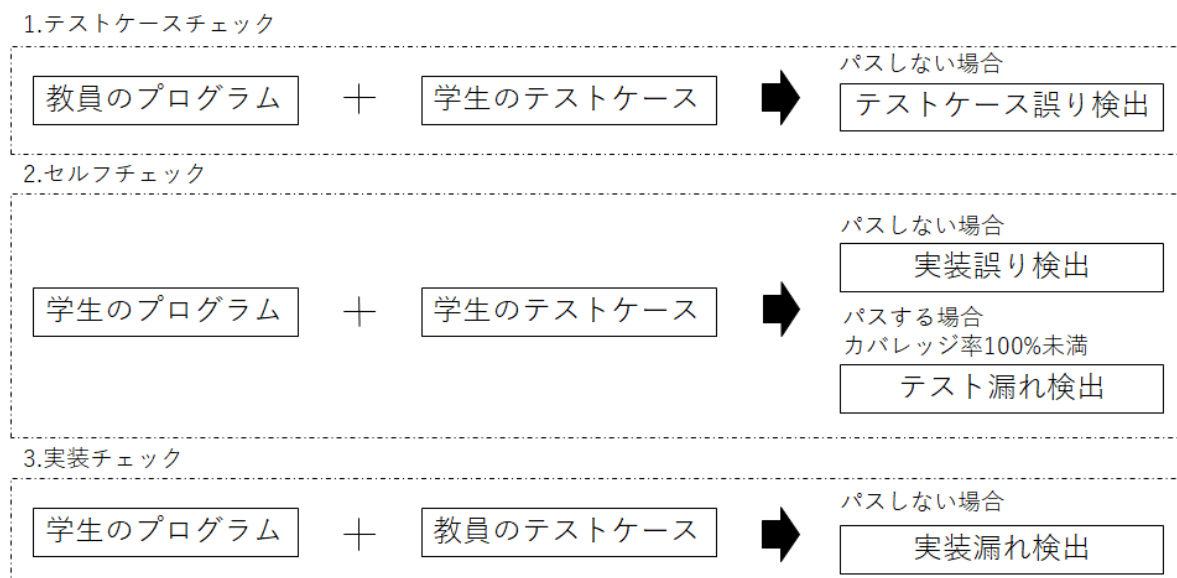


図 1.2 トリプルチェックテストによるコード判定原理

この手法では、テストケースの誤りを検出した際に、具体的にどのテストケースが不足しているのか学生に通知することができない。そこで、テストケースが適切であるかを判定する評価基準を用意して、評価基準に対する設計したテストケースの網羅率を求めることで評価を行い、不足しているテストケースがあった場合にアドバイスを文献 [3] の手法の拡張を考える。文献 [3] のテストケース評価システムの位置づけを

図 1.3 に示す。

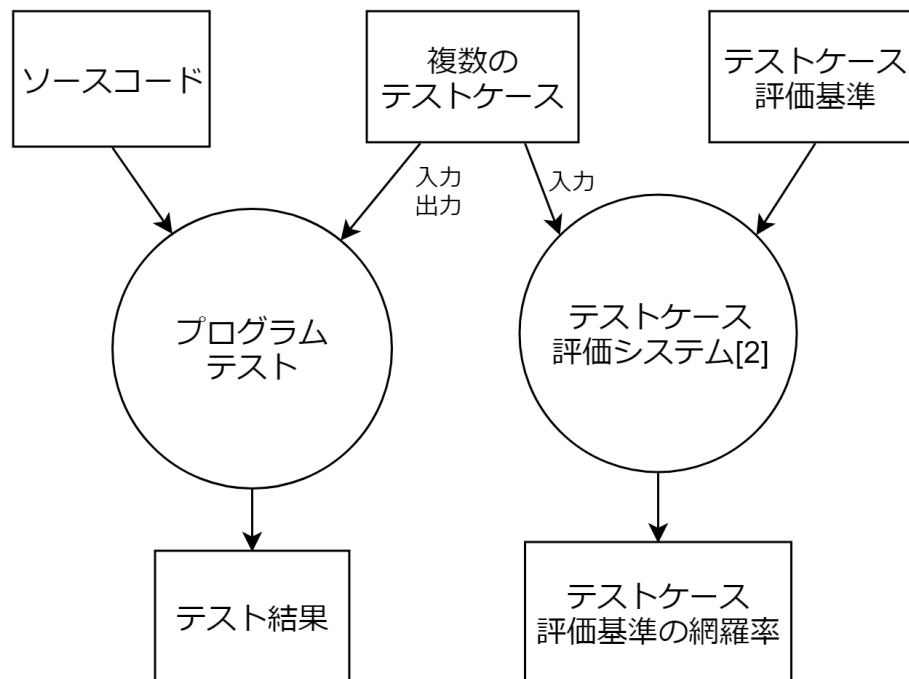


図 1.3 文献 [3] のシステムの位置づけ

1.2 目的

本研究では，評価基準作成時における教員の負担を減らすために，テストケースの評価基準を自動化を行うことを目的とする。プログラムテストとテストケース評価基準，本研究で開発するシステムの関係を図 1.4 に示す。

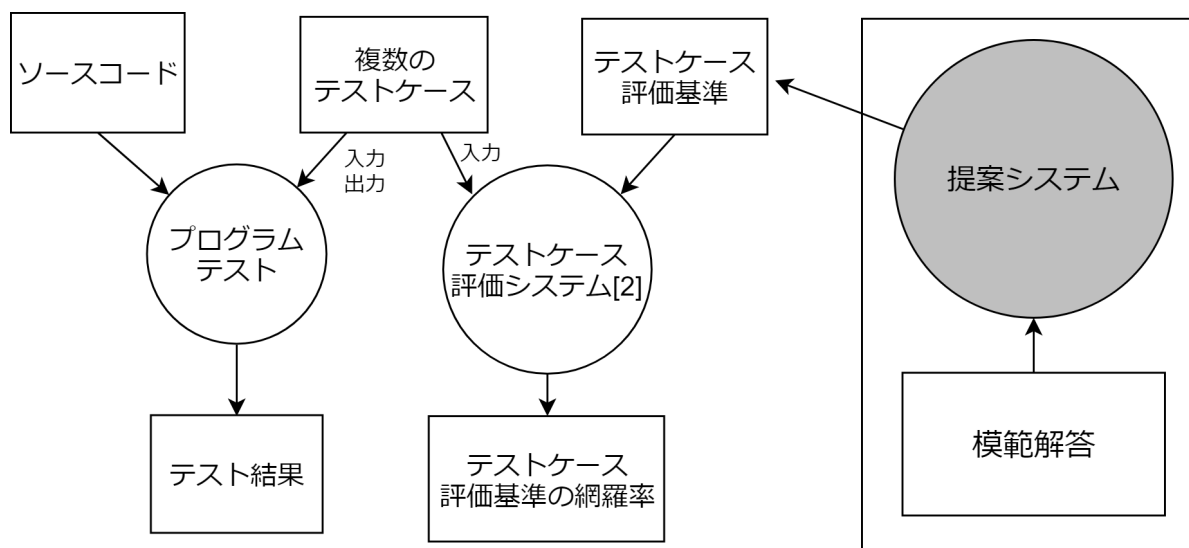


図 1.4 提案システムの位置づけ

1.3 論文の構成

以降，第 2 章では，関連研究としてテストケースの評価基準を用意してその網羅率によってテストケースを評価する手法を紹介する。第 3 章では，本研究で提案するテストケース評価基準の自動生成の手法についての説明および，評価基準生成システムを作成した手順について述べる。

第2章 プログラミング演習におけるテストケース評価システム

本章では文献 [3] で提案されているテストケース評価システムについて紹介する。テストケース評価システムでは、学生自身が適切なテストケースを設計できるようになるために、学生が作成したテストケースを評価してアドバイスをを行うシステムが提案されている。このシステムでは、教員が演習問題毎にテストケースの評価基準とテストケースが不足していた場合に表示するアドバイスを与える必要がある。演習問題毎にテストしなければならない値や入力データのデータ数が異なるため、評価基準は演習問題を分析した上で、入力データの構造を定義してから記述される。

2.1 テストケース評価システム

テストケース評価システムでは、教員があらかじめテストケース評価基準を作成し、学生が設計したテストケースが評価基準をどの程度パスできるかを判定することによって、テストケースの評価が行われる。評価基準をパスしない場合にアドバイスを表示することで、不足しているテストケースを学生に知らせる。テストケース評価システム [2] を使用する流れを図 2.1 に示す。テストケース評価システムにおいて、評価

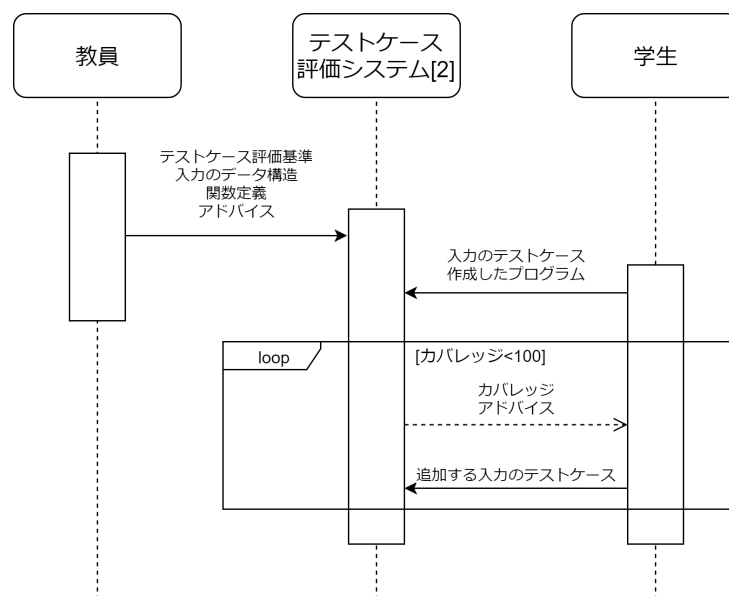


図 2.1 テストケース評価システム [2] のシーケンス図

基準は次のようにタブ区切り形式でテキストとして記述される。

評価基準の記述形式

評価基準番号 TAB 判定条件 TAB アドバイス

「判定条件」はテストケースにおける入力データが満たすべき条件で、その条件を満たすテストケースがなかったときに、学習者に「アドバイス」が表示される。同値クラスを評価するための基準が複数ある場合などは、同じ「評価基準番号」を指定することで、評価基準がグループ化できる。例えば、年齢（整数値）を入力して、20 歳以上は成年、20 歳未満は未成年と表示する課題の評価基準は、入力データを変数 `age` として、成年、未成年、エラーの場合について次のように記述される。

評価基準の記述例 1

- 1 TAB `age==20` TAB 成年の最低年齢
- 1 TAB `age>=20` TAB 成年の場合
- 2 TAB `age==19` TAB 未成年の最高年齢
- 2 TAB `age==0` TAB 0 歳
- 2 TAB `age>=0 && age<20` TAB 未成年の場合
- 3 TAB `age<0` TAB 年齢がマイナスの場合

学生が作成したテストケースの網羅率を、次の式で定義される評価基準のテストカバレッジにより提示する。

評価基準のテストカバレッジ（％）

学習者のテストケースがパスした評価基準グループ数 ÷
教員が記述した評価基準グループ数 × 100

判定条件の記述において、課題毎に入力データの型や個数が異なるため、テストケースの入力のデータ構造を定義し、そこで定義された変数を用いて判定条件が記述される。

2.2 入力のデータ構造の定義

データ構造は型と変数と出現回数から成る。入力のデータ構造の記述法はバックス・ナウア記法（BNF）に類した形式で設計された。入力のデータ構造の定義例を以下に示す。なお、`*` は 0 回以上の繰り返しを表し、`pint` は正の整数、`uint` は 0 と正の整数、`udouble` は 0 と正の実数の型である。

データ構造の定義例

(A) データ数 (n) を入力し、その個数分身体データを入力する場合

```
(pint n)
```

```
(pint id, string name, uint age, udouble height, udouble  
weight){n}
```

(B) ファイルの終端まで分身体データを入力する場合

```
(pint id, string name, uint age, udouble height, udouble  
weight){*}
```

2.3 関数定義

判定条件の記述に関数を用いることができる。複数の身体データを読み込んで、身長の最大値を表示する課題の場合、最大値が複数存在するテストの評価基準は、引数のリストの最大値の個数を返す関数 `countmax()` を定義して、次のように記述される。なお、実引数 `height` は全身長データのリストである。関数を実評価基準で用いる場合、教員は関数定義も与える必要がある。

関数を用いた評価基準の記述例

```
1 TAB countmax(height)>=2 TAB 身長の最大値が複数の場合
```

countmax 関数

```
function countmax($list){  
  $lmax = max($list);  
  return count(array_keys($list, $lmax));  
}
```

2.4 評価基準の具体例

入力が比較的複雑になる課題として、ボウリングの得点計算が例示されている。

ボウリングの点の計算の課題

ボウリングは、1 フレーム目から 10 フレーム目までの倒れたピンの数によって得点が決まる。フレームごとに倒したピンの数を読み込み、各フレームごとの得点を計算して表示すること。なお、ガーターとミスは 0 と表示するものとし、ダブルがあった場合はメッセージを表示する。

ボウリングのデータ構造

```
(uint thr1, uint thr2){9}
(uint last1, uint last2, uint last3)
```

ボウリングの評価基準

```
1 TAB thr1 == 10 TAB ストライクのスコア計算
2 TAB thr1 != 10 && thr1+thr2 == 10 TAB スペアのスコア計算
3 TAB last1 == 10 TAB ストライクの通常加算
3 TAB last1 != 10 && last1+last2 == 10 TAB スペアの通常加算
4 TAB double_strike(thr1) || all_strike([last1, last2]) TAB ダブル
4 TAB last_strike(thr1) == 1 && last1 == 10 TAB 最終 F をまたぐダブル
```

ストライクとスペアだった場合のスコア計算が正しいかをテストする必要がある（評価基準番号 1, 2）。最終フレームではストライク、スペアでも通常の加算になるので、それらのテストも必要である（評価基準番号 3）。評価基準番号 4 はダブルについてのテストである。ダブルがあれば true を返す関数 `double_strike()`、リストがすべて 10 ならば true を返す関数 `all_strike()`、リストの最後の要素から 10 が何回連続しているかを返す関数 `last_strike()` が定義されている。

2.5 システムの評価

参考書の例題のテストケースのデータ構造が評価システムに則した形式で記述できるか調査が行われ、実行時に入力データのデータ構造が変化する共用体などの問題以外ではデータ構造を記述できることが確認された。

また、評価システムにより、学生がテストケースを作成できるようになるのか確認をするために、プログラミングを学習済みの学生 7 人が評価システムを使ってボウリングの課題のテストケースを作成した。その結果最終的に 6 人の学生がカバレッジ 100% を達成したことが確認された。

2.6 システムの課題

テストケース評価システムでは、教員が問題文を分析した上で、評価基準を記述しているのですが、システムを運用する上で教員の負担が大きくなってしまっていることが考えられる。

第3章 テストケース評価基準の自動生成

本章では，テストケース評価基準を自動で生成する手法の概要とシステムを作成する手順を示す。参考書 [4] に記述されている見本のプログラムを参考に評価基準生成システムを作成する。

3.1 手法の概要

前章のテストケース評価システムでは，教員が問題文を分析をし，テストケース評価基準や入力データの構造などを事前に記述しなければならない。そこで，プログラミング演習では，通常のソフトウェア開発と異なり教員が模範解答を用意しているという点に着目し，提案システムでは，テストケース評価基準を生成する際に，問題文(仕様)を分析して記述するのではなく，模範解答(ソースプログラム)を解析し，入力変数と条件文を抽出することによって，テストケース評価基準を生成する。提案システムを用いてテストケース評価システムを使用する流れを図 3.1 に示す。また，入力された変数が正，負，0 である場合のテストケースがあるのかを確認するためのテストケース評価基準を生成する。さらに，main 関数以外の関数と関数の呼び出し部分を抽出し，テストケース評価基準とともに生成する。

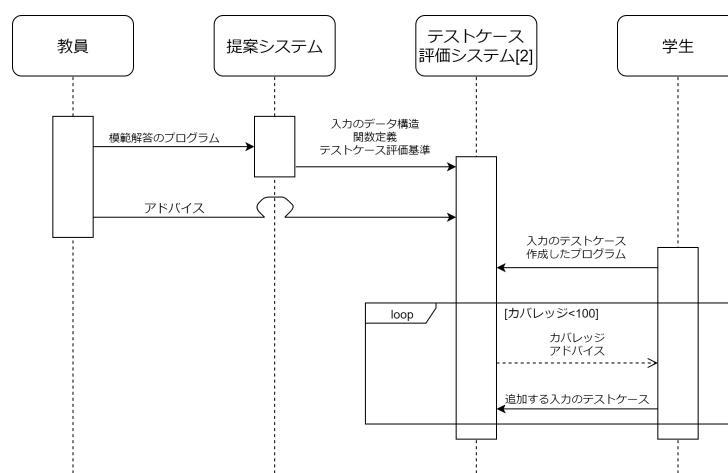


図 3.1 提案システムを用いたテストケース評価システム [2] のシーケンス図

3.2 入力データの型に基づく評価基準

まず、変数宣言から変数の型 t と変数名 x を抽出し保持する。標準入力命令

$$\text{scanf}(f, \&x)$$

が現れた場合に、変数 x の型を参照し、入力のデータ構造

$$(t\ x)$$

を生成する。変数 x の型 t が `int`, `double`, `float` のいずれかであった場合は

$$x > 0$$
$$x == 0$$
$$x < 0$$

を評価基準として記述する。以下に記述する入力された小数を表示するプログラムを用いて定義した通りの実相がされているかを確認する。

```
#include <stdio.h>

int main(void)
{
    double num;

    printf("小数を入力してください。 \n");

    scanf("%lf", &num);

    printf("%fが入力されました。 \n", num);

    return 0;
}
```

このプログラムでは、変数宣言部分から `double num` を取得し、標準入力命令

$$\text{scanf}("%lf", \&\text{num})$$

から `num` が入力変数であることを判定し、入力のデータ構造

$$\text{double num}$$

が生成される。また、変数 `num` は `double` 型なので、変数 `num` の判別式

```
num > 0
```

```
num == 0
```

```
num < 0
```

がテストケース評価基準として生成される。上記プログラムを入力としてシステムを実行した結果を以下に示す。

```
入力のデータ構造
(double num)
テストケース評価基準

入力変数の型から生成
num > 0
num == 0
num < 0
```

生成したテストケース評価基準をすべて満たすテストケースを入力とした場合の上記プログラムの実行結果を以下に示す。

```
小数を入力してください。
1.2
1.200000が入力されました。
小数を入力してください。
0
0.000000が入力されました。
小数を入力してください。
-0.6
-0.600000が入力されました。
```

3.3 条件文に基づく評価基準

3.3.1 条件文に入力変数がある場合

次に、条件判断文

```
if(c)
```

が現れた場合に条件文 *c* を抽出することでテストケース評価基準

c

を生成する。その際、分岐網羅をするために、条件文の否定

$!(c)$

もテストケース評価基準に含めて生成する。以下に記述する入力された整数が1の場合とそれ以外の場合で異なる文字を表示するプログラムを用いて定義した通りの実相がされているかを確認する。

```
#include <stdio.h>

int main(void)
{
    int res;

    printf("整数を入力してください。\\n");

    scanf("%d", &res);

    if (res == 1){
        printf("1が入力されました。\\n");
    }
    else{
        printf("1以外が入力されました。\\n");
    }

    return 0;
}
```

このプログラムでは、変数宣言部分から `int res` を取得し、標準入力命令

`scanf("%d", &res)`

から `res` が入力変数であることを判定し、入力データの構造

`int res`

が生成される。また、変数 `res` は `int` 型なので、変数 `res` の判別式

`res > 0`

`res == 0`

`res < 0`

がテストケース評価基準として生成される。また、条件判断文 `if` から条件文

`res == 1`

とその否定

`!(res == 1)`

がテストケース評価基準として生成される。上記プログラムを入力としてシステムを実行した結果を以下に示す。

```
入力のデータ構造
(int res)
テストケース評価基準

入力変数の型から生成
res > 0
res == 0
res < 0

条件文から生成
res == 1
!(res == 1)
```

生成したテストケース評価基準をすべて満たすテストケースを入力とした場合の上記プログラムの実行結果を以下に示す。

```
整数を入力してください。
1
1が入力されました。
整数を入力してください。
0
1以外が入力されました。
整数を入力してください。
-1
1以外が入力されました。
```

3.3.2 未定義の変数が条件文にある場合

さらに、関数宣言部分から関数名 x_f を取得し、条件文 c に関数名 x_f が使用されている場合は、関数の型を t_f 、引数の型を t_a 、引数の名前を x_a 、処理を p として、関数

$$\begin{array}{c} t_f \ x_f(t_a \ x_a) \\ \{ \\ \\ p \\ \} \end{array}$$

を生成する。以下に記述する入力された整数が偶数の場合と奇数の場合で異なる文字を表示するプログラムを用いて定義した通りの実相がされているかを確認する。

```

#include <stdio.h>

int even(int num)
{
    return num % 2;
}

int main(void)
{
    int num;

    printf("整数を入力してください。\\n");
    scanf("%d", &num);

    if (even(num) == 0)
    {
        printf("偶数です。\\n");
    }
    else
    {
        printf("奇数です。\\n");
    }
    return 0;
}

```

このプログラムでは、変数宣言部分から `int num` を取得し、標準入力命令

```
scanf("%d", &num)
```

から `num` が入力変数であることを判定し、入力データの構造

```
int num
```

が生成される。また、変数 `num` は `int` 型なので、変数 `num` の判別式

```
num > 0
```

```
num == 0
```

```
num < 0
```

がテストケース評価基準として生成される。また、条件判断文 `if` から条件文

```
even(num) == 1
```

とその否定

```
!(even(num) == 1)
```

がテストケース評価基準として生成される。さらに，条件文に関数 `even()` が使用されているので，関数 `even()` の定義

```
int even(int num)
{
    return num % 2;
}
```

が生成される。上記プログラムを入力としてシステムを実行した結果を以下に示す。

```
入力のデータ構造
(int num)
関数
int even(int num)
{
    return num % 2;
}

テストケース評価基準

入力変数の型から生成
num > 0
num == 0
num < 0

条件文から生成
even(num) == 1
!(even(num) == 1)
```

生成したテストケース評価基準をすべて満たすテストケースを入力とした場合の上記プログラムの実行結果を以下に示す。

```
整数を入力してください。
1
奇数です。
整数を入力してください。
0
偶数です。
整数を入力してください。
-3
奇数です。
整数を入力してください。
```

3.3.3 関数の戻り値を代入した変数が条件文にある場合

また、条件文 c に入力のない変数 x_u が使用されていて、その変数に関数 x_f を呼び出し、戻り値を代入している場合は、引数の型を t_a 、引数の名前を x_a として、関数呼び出し部分

$$x_u = x_f(t_a \ x_a)$$

を生成する。以下に記述する入力された整数が偶数の場合と奇数の場合で異なる文字を表示するプログラムを用いて定義した通りの実相がされているかを確認する。

```
#include <stdio.h>
int even(int num)
{
    return num % 2;
}
int main(void)
{
    int num;
    int mod2;
    printf("整数を入力してください。\\n");
    scanf("%d", &num);

    mod2 = even(num);
    if (mod2 == 0)
    {
        printf("偶数です。\\n");
    }
    else
    {
        printf("奇数です。\\n");
    }
    return 0;
}
```

このプログラムでは、変数宣言部分から `int num` を取得し、標準入力命令

`scanf("%d", &num)`

から `num` が入力変数であることを判定し、入力データの構造

`int num`

が生成される。また、変数 `num` は `int` 型なので、変数 `num` の判別式

`num > 0`

```
num == 0
```

```
num < 0
```

がテストケース評価基準として生成される。また，条件判断文 `if` から条件文

```
mod2 == 1
```

とその否定

```
!(mod2 == 1)
```

がテストケース評価基準として生成される。さらに，条件文に入力のない変数 `mod2` が使用されていて，`mod2` に対して関数 `even()` を呼び出し，戻り値を代入しているので，関数 `even()` の定義

```
int even(int num)
{
    return num % 2;
}
```

と関数呼び出し部

```
mod2 = even(num)
```

が生成される。上記プログラムを入力としてシステムを実行した結果を以下に示す。

```
入力のデータ構造
(int num)
関数呼び出し
mod2 = even(num);

関数
int even(int num)
{
    return num % 2;
}

テストケース評価基準

入力変数の型から生成
num > 0
num == 0
num < 0
```

```
条件文から生成  
mod2 == 0  
!(mod2 == 0)
```

第4章 プログラミング課題への適用

この章では、参考書 [4] の練習問題の模範解答から評価基準が生成できるかを検証する。参考書 [4] の 5 章（場合に応じた処理）までのサンプルコードや練習問題のうち、標準入力があるプログラムを対象に、作成したシステムによってテストケース評価基準の生成が可能か検証した。検証を行ったプログラムは 26 個であり、そのうちの 17 個で生成可能であった。生成できなかったプログラムは `getchar` で入力を行っているプログラムと、同じ変数に複数回入力しているプログラム、`switch` 文や条件演算子で場合分けをしているプログラムであった。また、繰り返し文や配列、関数を含むプログラムに対して評価基準を生成できるか検証を行った。

4.1 評価基準を生成できた問題

問題 1

身長を入力した後に体重を入力して順番に表示する。

```
#include <stdio.h>

int main(void)
{
    double height;
    double weight;

    printf("身長を入力してください。 \n");
    scanf("%lf", &height);

    printf("体重を入力してください。 \n");
    scanf("%lf", &weight);

    printf("身長は%fセンチです。 \n", height);
    printf("体重は%fキロです。 \n", weight);

    return 0;
}
```

このプログラムでは、変数宣言部分から `double height` と `double weight` を取得し、標準入力命令

```
scanf("%lf", &height)
```

```
scanf("%lf", &weight)
```

から `height` と `weight` が入力変数であることを判定し、入力のデータ構造

```
double height
```

```
double weight
```

が生成される。また、変数 `height` と変数 `weight` は `double` 型なので、変数 `height` と変数 `weight` の判別式

```
height > 0
```

```
height == 0
```

```
height < 0
```

```
weight > 0
```

```
weight == 0
```

```
weight < 0
```

がテストケース評価基準として生成される。

```
入力のデータ構造
(double height double weight)
```

```
テストケース評価基準
```

```
入力変数の型から生成
```

```
height > 0
height == 0
height < 0
weight > 0
weight == 0
weight < 0
```

```
身長を入力してください。
```

```
160
```

```
体重を入力してください。
```

```
50
```

```
身長は160.000000センチです。
```

```
体重は50.000000キロです。
```

```
身長を入力してください。
```

```
0
```

```
体重を入力してください。
```

```
0
```



```
身長は0.000000センチです。  
体重は0.000000キロです。  
身長を入力してください。  
-100  
体重を入力してください。  
-40  
身長は-100.000000センチです。  
体重は-40.000000キロです。
```

問題 2

キーボードから整数を 2 つ入力させ、値が大きいほうを表示する。

```
#include <stdio.h>  
int main(void)  
{  
    int num1;  
    int num2;  
    printf("2つの整数を入力してください。\\n");  
    scanf("%d", &num1);  
    scanf("%d", &num2);  
    if (num1 < num2)  
    {  
        printf("%dより%dのほうが大きい値です。\\n", num1  
            , num2);  
    }  
    else if (num1 > num2)  
    {  
        printf("%dより%dのほうが大きい値です。\\n", num2  
            , num1);  
    }  
    else  
    {  
        printf("2つの数は同じ値です。\\n");  
    }  
    return 0;  
}
```

このプログラムでは、変数宣言部分から `int num1` と `int num2` を取得し、標準入力命令

```
scanf("%d", &num1)
```

```
scanf("%d", &num2)
```

から `num1` と `num2` が入力変数であることを判定し、入力のデータ構造

```
int num1
```

```
int num2
```

が生成される。また、変数 `num1` と変数 `num2` は `int` 型なので、変数 `num1` と変数 `num2` の判別式

```
num1 > 0
num1 == 0
num1 < 0
num2 > 0
num2 == 0
num2 < 0
```

がテストケース評価基準として生成される。また、条件判断文 `if` から条件文

```
num1 < num2
num1 > num2
```

とそれぞれの否定

```
!(num1 < num2)
!(num1 > num2)
```

がテストケース評価基準として生成される。

```
入力のデータ構造
(int num1 int num2)

テストケース評価基準

入力変数の型から生成
num1 > 0
num1 == 0
num1 < 0
num2 > 0
num2 == 0
num2 < 0

条件文から生成
num1 < num2
!(num1 < num2)
num1 > num2
!(num1 > num2)
```

```
2つの整数を入力してください。
1
2
1より2のほうが大きい値です。
2つの整数を入力してください。
0
0
2つの数は同じ値です。
2つの整数を入力してください。
-3
-4
-4より-3のほうが大きい値です。
```

4.2 評価基準を生成できなかった問題

問題3

キーボードから5科目のテストの点数を入力させ、合計点と平均点を表示する。

```
#include <stdio.h>

int main(void)
{
    int sum = 0;
    int num = 0;

    printf("科目1の点数を入力してください。\\n");
    scanf("%d", &num);
    sum += num;

    printf("科目2の点数を入力してください。\\n");
    scanf("%d", &num);
    sum += num;

    printf("科目3の点数を入力してください。\\n");
    scanf("%d", &num);
    sum += num;

    printf("科目4の点数を入力してください。\\n");
    scanf("%d", &num);
    sum += num;

    printf("科目5の点数を入力してください。\\n");
    scanf("%d", &num);
    sum += num;

    printf("5科目の合計点は%d点です。\\n", sum);
    printf("5科目の平均点は%f点です。\\n", (double)sum /
        5);
```

```

        return 0;
    }

```

このプログラムでは，変数 `num` に対して入力が 5 回行われているので入力データの構造として

```
(int num){5}
```

が生成されなければならないが，現在は同じ変数が複数回入力される場合に回数を表示させることができず，

```
(int num)
```

のみ生成される。また，このプログラムには条件判断文 `if` がないため，入力データの型に基づく評価基準

```
num1 > 0
```

```
num1 == 0
```

```
num1 < 0
```

のみ生成されるが，5 科目の合計を求めるには，入力の値がすべて 0 以上でなければならないため，`num` が 5 回とも 0 以上である場合の評価基準が必要である。

問題 4

入力された整数が 1 なら A コース，それ以外なら B コースを選んだと表示する。

```

#include <stdio.h>

int main(void)
{
    int res;
    char ans;

    printf("何番目を選びますか？\n");
    printf("整数を入力してください。\\n");

    scanf("%d", &res);

    ans = (res==1) ? 'A' : 'B';

    printf("%cコースを選択しました。\\n", ans);

    return 0;
}

```

このプログラムでは、変数 `res` が 1 か否かの判定が行われているため、条件文

```
res == 1  
!(res == 1)
```

が生成されなければならないが、現在は条件判断文 `if` 以外からは条件文を抽出できないため、条件文に基づく評価基準が生成できない。

4.3 繰り返し文を含むプログラムでの検証

繰り返し文を含むプログラムでは、ループの開始時と終了時のテストが必要である。以下のプログラムでは、変数 `num` に 0 が代入された場合にループが終了するので、`num` が 0 の場合とそれ以外の場合の評価基準が生成されなければならない。しかし、現在のシステムでは繰り返し文から開始条件と終了条件を抽出することができないので、変数 `num` の型による評価基準のみ生成される。

```
#include <stdio.h>  
  
int main(void)  
{  
    int num = 0;  
    int sum = 0;  
  
    do{  
        printf("テストの点数を入力してください。(0で終了)\n");  
        scanf("%d", &num);  
        sum += num;  
    }while(num);  
  
    printf("テストの合計点は%d点です。 \n", sum);  
  
    return 0;  
}
```

4.4 配列を含むプログラムでの検証

配列を含むプログラムでは、変数宣言部から変数名を抽出するだけでは、入力の変数構造を十分に生成することができない。例えば、以下のプログラムでは、変数宣言部分が `int test[5]` となっているが、作られる配列は `int test[0]`, `int test[1]`, `int test[2]`, `int test[3]`, `int test[4]` である。つまり、変数宣言部分と標準入力命令で変数名が異なるため、現在のシステムでは入力変数として判定されない。配列を許容するシステムにするためには変数宣言時に配列が宣言されていた場合に変数を配列の長さ分用意する方法が考えられる。

```

#include <stdio.h>

int main(void)
{
    int test[5];

    printf("1人目のテストの点数を整数で入力してください\n");
    scanf("%d", &test[0]);
    printf("2人目のテストの点数を整数で入力してください\n");
    scanf("%d", &test[1]);
    printf("3人目のテストの点数を整数で入力してください\n");
    scanf("%d", &test[2]);
    printf("4人目のテストの点数を整数で入力してください\n");
    scanf("%d", &test[3]);
    printf("5人目のテストの点数を整数で入力してください\n");
    scanf("%d", &test[4]);

    for (int i = 0; i < 5; i++)
    {
        printf("%d番目の生徒は%d点です.\n", i + 1,
            test[i]);
    }
    return 0;
}

```

4.5 関数を含むプログラムでの検証

関数を含むプログラムでは、関数を呼び出して変数に代入していた場合に、関数呼び出し部分以降で変数に対して代入や加算などの処理を行っていないければ、評価基準を生成することができた。

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
/*BMIの定義*/
double BMI(double height, double weight)
{
    return weight / ((height * height) / 10000);
}
int main(void)
{
    double height;
    double weight;
    double bmi;

```

```

printf("身長を入力してください。\\n");
scanf("%lf", &height);
printf("体重を入力してください。\\n");
scanf("%lf", &weight);

if (height <= 0 || weight <= 0)
{
    printf("0より大きい値を入力してください。\\n");
    return -1;
}
bmi = BMI(height, weight);

printf("BMIは%fです。\\n", bmi);

if (bmi >= 25)
{
    printf("あなたは肥満です。\\n");
}
else if (bmi < 18.5)
{
    printf("あなたは低体重です。\\n");
}
else
{
    printf("あなたは普通体重です。\\n");
}
return 0;
}

```

身長と体重 2 つの値を入力して bmi を求め、肥満度を表示する上記のプログラムでは、変数宣言部分から `double height` と `double weight` を取得し、標準入力命令

```

scanf("%lf", &height)
scanf("%lf", &weight)

```

から `height` と `weight` が入力変数であることを判定し、入力のデータ構造

```

double height
double weight

```

が生成される。また、変数 `height` と変数 `weight` は `double` 型なので、変数 `height` と変数 `weight` の判別式

```

height > 0
height == 0
height < 0

```

```
weight > 0
```

```
weight == 0
```

```
weight < 0
```

がテストケース評価基準として生成される。また、条件判断文 `if` から条件文

```
height <= 0 || weight <= 0
```

```
bmi >= 25
```

```
bmi < 18.5
```

とその否定

```
!(height <= 0 || weight <= 0)
```

```
!(bmi >= 25)
```

```
!(bmi < 18.5)
```

がテストケース評価基準として生成される。さらに、条件文に入力のない変数 `bmi` が使用されていて、`bmi` に対して関数 `BMI()` を呼び出し、戻り値を代入しているので、関数 `BMI()` の定義

```
double BMI(double height, double weight)
```

```
{
```

```
    return weight / ((height * height) / 10000);
```

```
}
```

と関数呼び出し部

```
bmi = BMI(height, weight)
```

が生成される。上記プログラムを入力としてシステムを実行した結果を以下に示す。

```
入力のデータ構造
```

```
(double height double weight)
```

```
関数呼び出し
```

```
bmi = BMI(height, weight);
```

```
関数
```

```
double BMI(double height, double weight)
```



```
{  
    return weight / ((height * height) / 10000);  
}
```

テストケース評価基準

入力変数の型から生成

```
height > 0  
height == 0  
height < 0  
weight > 0  
weight == 0  
weight < 0
```

条件文から生成

```
height <= 0 || weight <= 0  
!(height <= 0 || weight <= 0)  
bmi >= 25  
!(bmi >= 25)  
bmi < 18.5  
!(bmi < 18.5)
```

生成したテストケース評価基準をすべて満たすテストケースを入力とした場合の上記プログラムの実行結果を以下に示す。

```
身長を入力してください。  
170  
体重を入力してください。  
60  
BMIは20.761246です。  
あなたは普通体重です。  
身長を入力してください。  
170  
体重を入力してください。  
50  
BMIは17.301038です。  
あなたは低体重です。  
身長を入力してください。  
160  
体重を入力してください。  
70  
BMIは27.343750です。  
あなたは肥満です。  
身長を入力してください。  
0  
体重を入力してください。  
0  
0より大きい値を入力してください。
```

第5章 おわりに

本研究では，学生が自身でテストケースの評価を行うために用いるテストケース評価基準を自動生成する方法を提案した。入力変数の型と変数名を取得することによって生成する，入力変数の型に基づくテストケース評価基準と，条件文を抽出することによって生成する，条件文に基づくテストケース評価基準の2種類のテストケース評価基準の生成を行った。教員が課題の模範解答（ソースプログラム）を用意すれば，問題文を分析することなく，入力のデータ構造や関数とともに評価基準を自動で生成することができた。提案システムを使うことにより，教員が用意するものが模範解答とテストケースが不足していた場合に表示するアドバイスのみとなるので，教員の負担を削減できたといえる。現在対応できていない配列や繰り返し文，関数を含むプログラムからの評価基準の自動生成が今後の課題である。

謝 辞

本研究を進めていく中で，多くの助言とご指摘をいただきました中村正樹准教授，
榊原一紀准教授に心より感謝の意を表します。また，合同ゼミにおいて，助言とご協
力をいただきました中村研究室，榊原研究室の皆様に感謝いたします。

参 考 文 献

- [1] 高橋直久, 丸山勝久: ソフトウェア工学, 森北出版, 2010.
- [2] 吉田英輔, 角川裕次: テスト駆動開発に基づくプログラミング学習支援システム : 初心者開発者のためのセルフトレーニングアーキテクチャ, 電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス, Vol.105, No.331, pp.27-32, 2005.
- [3] 蜂巢吉成, 小林悟, 吉田敦, 阿草清磁: プログラミング演習におけるテストケース評価システム, コンピュータソフトウェア第 34 巻第 4 号, pp.54-60, 2017.
- [4] 高橋麻奈: やさしい C, SB クリエイティブ, 2012.