B-028

記号実行を用いた等価ミュータント検出手法の提案
Equivalent Mutants Detection Using Symbolic Execution

上芝　貴也† 王　玮涛† 芳賀　博英†
Takaya Ueshiba Weitao Wang Hirohide Haga

1. はじめに

テストケースセットの品質を評価する手法であるミューテーション解析では，意図的にバグを埋め込んだプログラム（ミュータント）を生成し，テストケースがそのバグを検出できるかによって，テストケースの品質を評価する．しかしバグを埋め込む過程で，いかなるテストデータを与えても，元のプログラムと全く同じ動作をする等価ミュータントが生成されることがある．これは評価の妨げとなるので取り除く必要がある．本報告では記号実行という手法を用いて，等価ミュータントを取り除く手法を提案する．記号実行ではプログラムを値ではなく変数で実行し，出力は変数を含んだ式と条件の組で表される．ミュータントと元のプログラムについて記号実行を行い，その結果を比較することで等価ミュータントの検出を試みる．

2. ミューテーション解析

2.1 概要

ミューテーション解析はテストセットのバグ検出力を評価する手法である．テスト対象のプログラムに意図的にバグを埋め込み，そのバグをテストセットが検出できるかによって評価する．バグを埋め込む前のプログラムをオリジナル，埋め込んだ後をミュータントという．また，どのようなバグを埋め込むかという規則を，ミューテーションオペレータという．あるテストケースで，オリジナルとミュータントを実行したとき，それらの出力が異なれば，そのテストケースは埋め込まれたバグを検出したということになる．このとき，テストケースはミュータントを検出したという．

図 1 の例で考える．図の左側はオリジナルであり，2 つの整数を受け取って最大値を返すプログラムである．右はミュータントであり，4 行目の不等号の向きを逆にするというバグが埋め込まれているため，最小値を返すプログラムとなっている．これらのオリジナルとミュータントを使ってテストケースの品質を評価することを考える．

例えば，(1, 2)というテストケースでオリジナルとミュータントを実行する．オリジナルは最大値を返すので出力は 2 となる．一方ミュータントは最小値を返すので，出力は 1 となる．よって，同じ入力に対して出力が異なるため，テストケース(1, 2)はこのミュータントを検出したということになり，品質の良いテストケースと判定される．

次に，(1, 1)というテストケースを考える．このときオリジナルは最大値を返すので，出力は 1 となる．ミュータントは最小値を返すので，出力は同じく 1 となる．同じ入力(1, 1)に対してオリジナルとミュータントの出力が同じ 1 となったので，このテストケースはこのミュータントを検出できなかったということになり，品質の良くないテストケースと判定される．

† 同志社大学大学院理工学研究科,
Graduate School of Science and Engineering Doshisha University

```
int max(int x, int y)
{
  int z = x;
  if(x < y)
    z = y;
  return z;
}
```

```
int max(int x, int y)
{
  int z = x;
  if(x > y) /* “<” を “>” */
    z = y;  /*     に変更 */
  return z;
}
```

図 1　関数 max とそのミュータント

ミューテーション解析の結果として，ミューテーションスコアという，0 以上 1 以下の値が得られる．これは，生成された検出可能なミュータントがどのくらい検出されたか，すなわちテストセットの品質を示す値であり，次の式で表される．

ミューテーションスコア = 検出数 / 生成数

2.2 等価ミュータント

ミュータントの生成時，構文的にはオリジナルと異なるが，意味的には等価であるミュータントが作られることがある．これは等価ミュータントと呼ばれる．等価ミュータントは，任意のテストケースに対して出力がオリジナルと同じになる．つまり，どんなテストケースもそのミュータントを検出できない．したがって，ミューテーションスコアを正しく算出することができず，テストセットの品質を評価する際の妨げとなるので，除去する必要がある．

図 2 は関数 max の等価ミュータントの一例である．4 行目の x を z に変更するというバグを埋め込んでいる．しかし，直前の 3 行目で z には x が代入されているので，4 行目における x と z の値は同じである．したがって，このミュータントはオリジナルと全く同じように振る舞うので，等価ミュータントである．

```
int max(int x, int y)
{
  int z = x;
  if(x < y)
    z = y;
  return z;
}
```

```
int max(int x, int y)
{
  int z = x;
  if(z < y) /* “x” を “z” */
    z = y;  /*     に変更 */
  return z;
}
```

図 2　関数 max とその等価ミュータント

3. 提案手法

以前の研究では，ミュータントの変更箇所が実行されるか否かを調べることで等価ミュータントの自動検出を行い，結果は約 27%であった[1]．今回は，それに加えて記号実行を用いて検出を試みた．なお，1 つのソースコードに対し，

変更は 1 カ所とし，ミューテーション解析の対象とするプログラムは C 言語としている．

3.1 記号実行を用いた等価ミュータントの検出

等価ミュータントは，任意の入力に対してオリジナルと同じ出力を返す．また，記号実行は任意の入力に対する出力を，変数を含んだ式と条件の組として表現する．したがって，オリジナルとミュータントそれぞれに対して記号実行を行うことで，任意の入力に対する出力を得られ，それらが全く同じならば，任意の入力に対してオリジナルとミュータントの出力が同じであるということなので，等価ミュータントの判定が可能であると考えられる．本研究ではこれを行うツールを作成した．

3.2 記号実行の概要

記号実行とは，関数を実行するときに具体的な値を用いずに，記号のまま実行する手法である．通常の実行では入力は具体的な値であり，出力もまた具体的な値となる．一方，記号実行では入力は変数であり，出力は変数を含んだ式と条件の組となる．

図 2 左側の例で考える．具体的な値による実行では，例えば入力を(1, 2)とすると，出力は 2 となる．つまり，式で表すと次のようになる．

```
max(1, 2) = 2
```

一方，記号実行では，入力は変数となり，出力は変数を含んだ式と条件の組となる．式で表すと次のようになる．

```
max(x, y) = x  (x >= y)
            y  (x < y)
```

このように，記号実行を用いることで，関数の挙動を調べることができる．

4. 評価実験

既存のツール[1]に記号実行に基づく手法を組み合わせることにより，どの程度等価ミュータントの検出精度が向上するかを評価するために，評価実験を行った．実験には 2 種類のプログラムを用いた．一つは，int 型の値を 3 つとり，中央値を返す mid であり，もう一つは，int 型の値を 3 つとり，それらによって作られる三角形の型を返す triangletype である．それぞれのプログラムについて，ミュータントを生成し，等価ミュータントを人間の手で調べたのち，ツールによる自動検出を行った．なお，利用したミューテーションオペレータは**表 1** に示す．これらは，以前の研究[1]で用いたものと同一のオペレータである．

実験結果は**表 2** のとおりである．平均で約 70 % の等価ミュータントが検出された．また，等価ではないミュータントを，等価ミュータントであると判断してしまうような誤検出はなかった．

表 1 使用したミューテーションオペレータ

名前	意味
ABS	Absolute value insertion
AOR	Arithmetic operator replacement
ROR	Relational operator replacement
LOR	Logical operator replacement
ASR	Assignment operator replacement
UOI	Unary operator insertion
SVR	Scalar variable replacement

表 2 実験結果

名前	生成数	等価数	検出数	検出率
mid	151	15	9	60%
triangletype	444	44	36	81%

5. 考察

記号実行を用いたことにより，等価ミュータントの検出率は平均で約 70%となり，以前作成したツール[1]の 27%よりも多くの等価ミュータントを検出できるようになった．

しかし，依然として検出されていない等価ミュータントがある．例として，**図 3** のような等価ミュータントが挙げられる． このミュータントは，不等号<を<=に変更するバグが埋め込まれている．オリジナルとミュータントが異なる経路を通るためには，x<y と x<=y で出力が異なる必要があるので，x==y である必要がある．しかし，x==y のとき，オリジナルとミュータントは出力が同じになってしまうので，これは等価ミュータントである．このような事例は，単純に記号実行しただけでは検出はできず，改良の余地がある．

```
int max(int x, int y)
{
  int z = x;
  if(x < y)
    z = y;
  return z;
}
```

```
int max(int x, int y)
{
  int z = x;
  if(x<=y) /* “<” を “<=” */
    z = y; /*     に変更  */
  return z;
}
```

図 3 関数 max と未検出の等価ミュータント

また，今回評価実験に用いたプログラムは if 文や代入文を含んだ単純なものである．今後はより複雑なプログラム，例えば繰り返し文や配列などを含んだものでも実験を行い，今回作成したツールの有用性を詳しく評価する必要がある．また，ミューテーションオペレータに関しても，より多くの種類を使って実験をする必要がある．

6. おわりに

等価ミュータントはミューテーション解析によるテストセット品質評価の妨げとなるため，取り除く必要がある．従来は人間が取り除いていたため，時間がかかったり，誤検出してしまったりする問題があった．この論文では，記号実行を用いた等価ミュータント検出ツールを作成した．評価実験では，生成された等価ミュータントのうち，平均で約 70%が検出された．これにより，人手による等価ミュータントの除去を大きく削減することが可能となった．今後はこのツールの有用性を詳しく評価するために，より多くのプログラムで評価を行う必要がある．また，今回評価実験に用いたプログラムに対しても，今回利用しなかったミューテーションオペレータを使うことで詳細に評価する必要がある．

参考文献

[1] 上芝 貴也, 末広 暁久, 芳賀 博英, “ミューテーション解析における等価ミュータント検出機能の実装”, 情報科学技術フォーラム, 11th, 241-242 (2012).