4R−1　変更を受けたソフトウェアを対象とするテストケース生成

古谷　信俊　　花岡　晃浩　　深澤　良彰　　門倉　敏夫
早稲田大学　理工学部

1　はじめに

ソフトウェアの開発やその保守において、ソフトウェアの変更は頻繁に行われている。部分的な変更を受けたソフトウェアに対し、初めからテストをやり直すことは経済的ではない。従って、変更の影響を受ける箇所のみをテストする手法が重要である。

本研究の目的は、変更が行われたソフトウェアに対し、最小限のコストで必要かつ良質のテストを行うテストケースを生成することである。ここでテストとは、変更後のプログラムが仕様を正しく実現していることを調べることを意味している。

テストにはブラックボックス、ホワイトボックスの2つの手法があるが、どちらも一長一短があり、完全ではない。本研究ではテストをより完全にするために両方の手法を取り入れている。

2　システムの概要

本システムの概要を図1に示す。

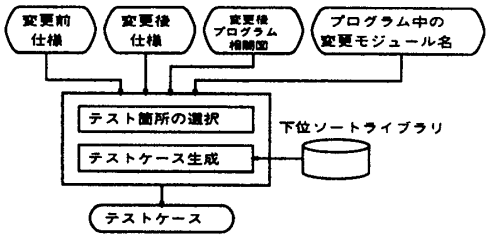

図 1: システム概要図

一般に仕様の変更とインプリメントの変更は必ずしも1対1に対応しない。従って、テストすべき箇所を特定するためには仕様とプログラムの両方の変更情報が必要となる(図1)。

本システムに入力されるテスト対象の仕様は、半構成的代数仕様[1]として記述されているものとする。半構成的仕様記述は、以下のような特徴を持つ。

- 公理部が合流性、停止性を満たす右向きの項書き換え規則であること。
- 公理は左辺の初めに登場する演算記号の定義と解釈できること。

仕様中に登場する演算記号は、プログラム中で手続きや関数などのモジュールとして定義されているものとする。また、仕様で用いる各ソート(型)はプログラミング言語で用意されているものか、ユーザ定義されているものとしている。

システムに入力されるプログラム情報は、変更後のプログラムにおけるモジュール相関図と、直接変更を受けたモジュール名の二つである。ただし、モジュール相関図はプログラムから自動生成される。

本システムの内部処理は、テストすべき箇所の選択と、選択された箇所におけるテストケース生成の2段階から成っている。本研究では公理ベースのテストケース生成法を採用するため、テスト箇所の選択はテストすべき公理の選択となる。テストケース生成系では、テストすべき公理中の変数へ、変数を含まない項を割り当てた形のテストケースを生成し、出力する。

3　テストケース生成戦略

3.1　テスト箇所の選択

(1)　仕様の情報による選択

本システムで用いる仕様は半構成的であるため、公理は左辺の最左の演算記号の機能を定義するものと見なすことができる。従って、公理が変更されたということはその演算記号の定義が変更されたことを意味する。これより、変更された公理の左辺の最左の演算記号を含む公理が変更の影響を直接受けると考えることができる。また、こうして選ばれた公理によって定義されている演算記号を含む公理もテスト対象公理とする。これは仕様とプログラムの演算間の依存関係が必ずしも一致していないためである。

以上の操作を、新しくテスト対象公理としてつけ加えるものがなくなるまで繰り返す。

(2)　プログラムの情報による選択

あるモジュールに手を加えた場合、それを呼び出しているモジュールは影響を受ける。本システムでは、変更されたモジュールから相関図を上にたどって行き、仕様の演算記号と対応しているモジュールを全て選ぶ。最後に、それらのモジュールに対応している演算記号を含む公理をテスト対象公理とする。

3.2　テストケース生成

選択された公理について、そのテストケースを生成する。本研究では L.Bouge の方法[2]に手を加えたものを用いている。

本研究で改良した点について以下に述べる。

Test Case Generation for Changed Software
Nobutoshi FURUYA, Akihiro HANAOKA,
Yoshiaki FUKAZAWA, Toshio KADOKURA
School of Science and Engineering, Waseda University.

(1) 組合せ条件部のテストケース

次の公理は、ボールゲームのスコア管理の仕様の一部であり、15点先取のルールをデュース制に変更したことを示している。

```
Iswin(Red,Score(r,w)) = if r==15
                            then true
                            else false
                        endif
                    ⇓
Iswin(Red,Score(r,w)) = if r≥ 15 and r≥ w+2
                            then true
                            else false
                        endif
```

Bouge の方法では、IF 文による分岐の両側を網羅するようにテストケースを生成する。従って、変更後の公理に対して生成されたテストケースにおいて、変数 r、w への値の割当ては、例えば (r,w) = {(15,0),(0,0)} となる。しかし、これは変更前の公理に対するテストケースと同じであり、デュース制によって変更された部分のテストにはなっていない。これより本システムでは、条件部が複数の条件の複合形である場合には、各条件の結果の組合せを全て網羅するようにして、テストケースの範囲を広げた。これは、ホワイトボックステストの手法で、分岐網羅から、条件網羅に変更したことに相当する。上の例では、本システムの生成するテストケースは、(r,w) = {(15,0),(15,13),(2,0),(0,0)} という割当てを含む。ただし、条件の組合せによっては解が得られない場合がある。このため、演算記号の最大のネスト数をあらかじめ与えておき、その範囲内で解が見つからない場合は解なしとして計算を停止させる。

本手法では、条件部の細かい条件の数が増えるにつれ組合せの数が指数的に増加する。しかし、記述の経験から、抽象度の高い仕様において条件部がそれほど複雑になることは少なく、たとえテストケースの数が多少増えてもテストケース生成からテスト判定のプロセスまでを自動化することにより十分対処できる。

(2) 低レベルソートのテストケース

Bouge の戦略では、IF 条件部に関与しない下位ソート変数へ割り当てる値をランダムに選択している。本システムでは下位ソートライブラリを使って、あらかじめ用意された各ソートの境界値を変数に割り当てる。扱うデータの境界値付近でエラーの発生する頻度が高いため、ランダムな値の割当てに較べてより有効なテストが期待できる。

(3) 両辺がブール型を返す公理について

たとえば、f(x) = x / 2 という等式的公理に対して、本システムでは下位ソートの情報より (x) = {0,32767} というテストケースを生成する。もし、f(x) = x == 1 という公理を上と同様な公理として扱うと、同じテストケースが生成されることとなる。この場合、公理の両辺が真になるテストケースが含まれていないため不十分である。従って、両辺が真、偽以外のブール型の演算記号から形成されている場合は、特別に真、偽の両方の結果が得られるテストケースを生成する。

4 評価

プログラム中でテストすべき箇所を特定し、それらの箇所について生成テストケースが満たす分岐カバレージを調べ、テストケースの有効性を示す。テストすべき箇所はプログラム中の直接変更を加えた部分とした。

表1に本システムが生成したテストケースと、Bouge のオリジナルの方法によって生成したテストケースによるカバレージをそれぞれ示す。

表 1: 各テストケースによるカバレージ

仕様例	Bouge	本システム
ボールゲーム スコア	11/13 (85%)	13/13 (100%)
エディタ	7/7 (100%)	7/7 (100%)
図書館 データベース	17/18 (94%)	18/18 (100%)

表1より、組合せ条件部に対する処理の改良による効果は明らかである。また、上記の基準で選んだ箇所に関して本システムでは 100% の分岐カバレージを満たしていることも評価できる。

次に、変更後のソフトウェア全体に対するテストケースを生成した場合に比べ、テストケースの数をどの程度削減することができたかを調べ、表2に示す。ただし、表中の減少率とは、変更後のソフトウェア全体を対象として生成したテストケースの数に対する、本システムが生成したテストケースの数の割合を意味している。

表 2: テストケース数の減少率

仕様例	減少率
ボールゲーム スコア	14/16 (88%)
エディタ	91/247 (37%)
図書館 データベース	60/248 (24%)

5 おわりに

本稿では、変更が行われたソフトウェアに対し、良質かつ無駄の少ないテストケースを生成する手法を与えた。

今後は、テストの自動実行系、テスト結果の自動判定系も含めた統合的なテスト環境を目指していく予定である。

参考文献

[1] Ivo Van Horebeek, and Johan Lewi, *"Algebraic Specifications in Software Engineering,"* Spring-Verlag, 1989.

[2] L.Bouge, N.Choquet, L.Fribourg, and M.-C.Gaudel, *"Test Sets Generation from Algebraic Specifications Using Logic Programming,"* J.System and Software 6, pp.343-360, 1986.