1J-03

形式仕様に基づくテストケース生成法の評価
Evaluation of a Formal-Specification-Based Test Case Generation Method

福岡　真吾
Shingo Fukuoka
法政大学情報科学部コンピュータ科学科
E-mail: shingo.fukuoka.8b@stu.hosei.ac.jp

Abstract

Test case generation methods are extremely important for program testing. A formal specification-based test case generation method known as SOFL-SBTCG[1] has been proposed, but its effectiveness has not been properly evaluated. In this paper, we describe how the method is evaluated by means of comparison with pairwise testing and random testing that are both commonly used in practice. Our discussion focuses on the issues of how a medical system is prepared as the target program system for testing, how test cases are generated using all of the three test case generation methods respectively, and how the test results are analyzed to evaluate the effectiveness of the SOFL-SBTCG. The result of the analysis indicates that the SOFL-SBTCG is superior to both pairwise and random testing in certain circumstances.

1. まえがき

ソフトウェアテストは，ソフトウェアエラーを発見する重要な技術である．よく使われるテストケースの有効性を評価する基準は，コードのパスカバレッジ(コード網羅率)である．もちろん，テストケース数を増やせば，コードカバレッジも増加する．しかし，テストケース数が増加すれば同時にテストにかかるコストも増加する．そのためテストケース生成法には，より少ない項目数で高いコードカバレッジを達成できる事が求められる．そこで本論の目的は，新しいテストケース生成法である Test Case Generation Based on Formal Specifications （SOFL-SBTCG）[1]を評価することである．評価方法は SOFL[2]を用いて開発したソフトウェアに対して，同じくブラックボックステストであり，実際の開発でも用いられている Pairwise [3] 法および Random testing[4]との比較を行う．

2. 設計

本システムの機能は大きく分けると，「ユーザー情報の操作」「センサの情報の操作」「ユーザーの診断」「数学的計算」の 4 つである．データリソースは，「ユーザー情報」「センサ情報」「センサ基準値表」「診断テーブル」の 4 つである．制約は，「ユーザーの不具合の分析は 1 度センサで計測を行わないとできない。」「過去のセンサ情報は 10 回まで保存され、以降古いものから上書きされる。」「ユーザーの年齢は半角数字以外受け付けない。」の 3 つである．

このシステムを SOFL に基づき，Informal Specification, Formal Specification に分けて設計し，実装は開発言語は C#を使い，開発環境は Visual Studio2010 で開発を行った．

3. テスト方法

新しいブラックボックステストの手法の 1 つである，SOFL-SBTCG [1]を評価するため，本ソフトウェアを 3 種類の方法でテストケースを生成しテストする．生成したテストケースを NUnit を用いてテストを行い，コードカバレッジの測定は PartCover を利用した．

3.1. SOFL-SBTCG

このテストを行うためには，SOFLの仕様書から FSF(Functional Scenario Forrm)を生成する必要があるため，3つのステップを踏みテストケースを生成する． FSFは，“Definition 1:
Let $S_{Post} \equiv (C_1 \wedge D_1) \vee (C_2 \wedge D_2) \ldots (C_n \wedge D_n)$, where each $C_i (i \in \{1,..,n\})$ is a predicate called a “guard condition” that contains no output variable in S_{ov} and $\forall_{i,j \in 1,..,n} \cdot i \neq j \Rightarrow C_i \wedge C_j = \text{false}$; D_i a “defining condition” that contains at least one output variable in S_{ov} but no guard condition. Then, a (functional) scenario f_s of S is a conjunction $S_{pre}{\sim} \wedge (C_i \wedge D_i)$, and the expression $(S_{pre}{\sim} \wedge C_1 \wedge D_1) \vee (S_{pre}{\sim} \wedge C_2 \wedge D_2) \vee \cdots \vee (S_{pre}{\sim} \wedge C_n \wedge D_n)$ is called a functional scenario form (FSF) of S.”[1]
と定義される形である．

1. 最初に，対象のプロセスの post condition を Disjunctive Normal Form(積和標準形)に変換し．pre condition と組み合わせ，FSF を生成する．
2. 次に，FSF を，
$f_{1=} S_{pre}{\sim} \wedge C_1 \wedge D_1$，$f_{2=} S_{pre}{\sim} \wedge C_2 \wedge D_2$，・・，
$f_{n=} S_{pre}{\sim} \wedge C_n \wedge D_n$
という形に変換する．つまり，conjunction に分解する．このときの 1 つ 1 つのf_nが，Functional Senario の形になっている．
3. 最後に，f_nを true にすることができる入力でテストケースを生成するというのがこのテスト方法である．

Supervisor: Prof. Shaoying Liu

Copyright ©2016 Information Processing Society of Japan.
All Rights Reserved.

```
process InitializeUserInformation(password:
string)initialize_message: string
ext wr user_info:UserInformation
pre user_info.password = password
post user_info = mk_UserInformation(nil,nil) and
initialize_message = "ユーザー情報が初期化されました"
end_process;
```

図 1.　Formal Specification 上のユーザー情報の初期化

例として，図 1 の仕様書からテストケースを生成する．DNF への変換は既に，完了しているので省略し FSF の生成から始める．

```
user_info = mk_UserInformation(nil,nil) and
initialize_message = "ユーザー情報が初期化されました"
```

図 2.　ユーザー情報の初期化プロセスの FSF

次に，FS を生成する．

今回の process は，if 文や case 文，or などが存在しないため，1 つの FS のみ生成可能である．

```
user_info.password = password  and
user_info = mk_UserInformation(nil,nil) and
initialize_message = "ユーザー情報が初期化されました"
```

図 3.　ユーザー情報の初期化の FS

最後に，この FS を基にテストケースを生成したものが以下の表 1 である．

表 1.　ユーザー情報の初期化のテストケース(FSF)

password	user_info
“correct_password”	{“Liu”,”男”,55, ”correct_password”}

3.2. Pairwise Testing (All-pair Testing)

今回，Equivalent Partition(同値分割法)を用いて，SOFL の仕様書から各 factor の level を生成した．ここで仕様書を分割できると考えたのは 3 つの記述である．

1. pre-condition
2. 入力変数の型による制約
3. post-condition 内の条件分岐

例として，図 1 の仕様書からテストケースを生成した．

表 2.ユーザー情報の初期化のテストケース(Pairwise)

password	user_info
“correct_password”	{“Liu”,”男”,55, ”correct_password”}
“wrong_password”	{“Liu”,”男”,55, ”correct_password”}

3.3. Random Testing

今回は 1 つの機能につき，3 つのテストケースを生成した．例として，図 1 の仕様書のテストケースを生成したのが以下の表 3 である．

表 3.ユーザー情報の初期化のテストケース(Random)

password	user_info
"QtK8ETMlxAfqS"	{"7RCOuR4FaJbr","男",666843723,"RJprTisxUpi"}
"jbaFqn"	{"9kz1ZcMKMT5ul","女",1570139137,"szZ2X"}
"Q"	{"tM1FBxx7","女",1164129559,"xRChTL7DxDAnpp"}

4. テスト結果

表 4.システム全体のテスト結果

	SOFL-SBTCG	Pairwise	Random
テストケース数	34	67	72
コードカバレッジ (Initialize UserInformation)	78%	79%	62%

5. 考察

今回のテスト結果としては，SOFL-SBTCG は，テストケースの生成数が最も少ないが，高いコードカバレッジを測定している．Pairwise Testing は，テストケースの生成数は多いが，最も高いコードカバレッジを測定した．Random Testing は，最も多くテストケースを生成しているにもかかわらず，最も低いコードカバレッジを計測したという結果を得た．

SOFL-SBTCG は，プログラムの持つべき機能を保証することができるため，テストケースの少なさに比べて高いコードカバレッジを計測することができる．しかし，この方法では仕様の pre condition を満たさないケースをカバーしていない．そのため，プログラム内のエラー処理のパスを実行することができない．

他の評価の仕方として，テスト項目数を増やした際のそれぞれのテストのカバレッジの増加曲線，他のテスト手法との比較があればより，この新しいブラックボックステスト手法の評価を深めることができるだろう．

文　献

[1] Shaoying Liu, Shin Nakajima , A Decompositional Approach to Automatic Test, Secure Software Integration and Reliability Improvement (SSIRI), pp 148-155, IEEE International Conference on Secure Software Integration and Reliability Improvement, 2010

[2] Shaoying Liu, Formal Engineering for Industrial Software Development Using the SOFL Method, Springer-Verlag, 2004

[3] James Bach, Patrick J. Schroeder, Pairwise Testing: A Best Practice That Isn’t, 2004

[4] Alessandro Orso, Gregg Rothermel, Software testing: a research travelogue (2000–2014), Proceedings of the on Future of Software Engineering, pp 117-132, 2014

[5] 「家庭の医学」, http://health.goo.ne.jp/medical,

Copyright ©2016 Information Processing Society of Japan.
All Rights Reserved.