4J-05

形式仕様に基づくテストケースの自動生成とテスト結果の自動評価

池田　逸人 †
法政大学情報科学研究科 †

劉　少英 ‡
法政大学情報科学研究科 ‡

1. まえがき

ソフトウェアは，交通システム，医療機器，ATM 等の様々な分野で利用されている．これらのソフトウェアは，プログラムのエラーやバグでシステムが停止することは合ってはならない．よって信頼性を確保するために，形式手法を用いた開発やテストを行うことで信頼性のあるソフトウェアが開発できる [1]．しかしながら，テストや評価に時間がかかるという問題点が残っている．

この問題を解決するため，形式仕様から自動的にテストケース生成とテスト結果の評価するツールを開発した．本論文では，SOFL 形式仕様 [2] の入力変数，出力変数，事前条件と事後条件によって，テストケース生成を自動生成する方法と自動化について述べる．

2. 形式仕様によるテストケース生成方法

SOFL 形式仕様の process で定義された入力変数の値の生成を試みる．process では，入力変数 S_{iv}，出力変数 S_{io}，事前条件 S_{pre} と事後条件 S_{post} を定義する．S_{pre} と S_{post} は，述語論理によって定義される．この S_{iv} の具体的な値がプログラムにに与えるテストケースである．事後条件 S_{post} は，$S_{post} \equiv (C_1 \wedge D_1) \vee (C_2 \wedge D_2) \vee \cdots \vee (C_n \wedge D_n)$　と表現できる．この条件の中で，$C_i (i \in \{1, ..., n\})$ を”Guard Condition(GC)”と定義し，C_i は出力変数を含まない条件である．$D_i (i \in \{1, ..., n\})$ を”Defining-Condition(DC)”と定義し，D_i は出力変数を含む条件である．テストケースの生成や評価を行うためには，S_{pre} と S_{post} から Functional Scenario Form(FSF) に変換する [3]．F_i に対してそれぞれ S_{iv} の具体的な値がテストケース T_i である．T_i が満たす必要がある条件は，FSF から D_i を除いた Test Condition Form である．

定義 1 (Functional Scenario Form) *Functional Scenario Form* は，$(S_{pre} \wedge C_1 \wedge D_1) \vee (S_{pre} \wedge C_2 \wedge D_2) \vee \cdots \vee (S_{pre} \wedge C_n \wedge D_n)$ と定義する．また，$(S_{pre} \wedge C_i \wedge D_i)$ を *Functional Scenario(FS)* と呼ぶ．

定義 2 (Test Condition Form) $(S_{pre} \wedge C_1) \vee (S_{pre} \wedge C_2) \vee \cdots \vee (S_{pre} \wedge C_n)$ を *Test Condition Form* と定義する．また，$(S_{pre} \wedge C_i \wedge D_i)$ を *Test Condition(TC)* と呼ぶ．

Automatic Test Case Generation and Test Result Evaluation based on Formal Specifications
†Hayato Ikeda ・ Hosei Univ.
‡Shaoying Liu ・ Hosei Univ

3. テストケース自動生成方法

この章では，章 2 で示した方法を自動で行うアルゴリズムについて説明する．

3.1. TC からテストケース生成

TC からテストケースを求めるアルゴリズムを疑似コード 1 に示す．まず，原子論理式 P_1 に対して GenerateTestCase によってテストケース生成を行う．節 3.3. にて原子論理式からテストケース自動生成する方法について説明する．そして求めた S_{iv} は $P_2 \wedge \cdots \wedge P_n$ に代入する．P_n までテストケースを生成したら，TC が真であるかどうかを判断する．TC が真であれば，TC に含まれる S_{iv} は条件を満たすテストケースである．偽であれば生成は失敗であり，結合法則を利用して原子論理式 $P_1, ..., P_n$ の順番を入れ替える．テストケースを生成する順番を変えることで，成功することがあるためである．入れ替える TC の順番は n!通り存在し，TC が真になるまでこの方法を適用する．疑似コード 1 の Permutation は，BaseTC に対して原子論理式の順番を入れ替えた i 通り目の TC を取得する関数である．それでもテストケースを生成できない場合は，テストケースを ”nil ”にする．

Algorithm 1 Generate test case from Test Condition

```
1: (TC = P1 ∧ P2 ∧ ··· ∧ Pn)
2: BaseTC = TC
3: for i ← 1 to n! do
4:     for j ← 1 to n do
5:         Siv ⇐GenerateTestCase(Pj)
6:         TC に Siv を代入する
7:     end for
8:     if TC = true then
9:         Siv がテストケースとなる
10:        break;
11:    else
12:        TC ⇐ Permutation(BaseTC, i)
13:    end if
14: end for
```

3.2. 原子論理式からテストケース生成

原子論理式 P には複数の演算子が含まれているため自動生成が困難である．演算子が 1 つの式に変換して，テストケース生成を試みる．

3.2.1. 演算子が 1 つの式に変換

優先度の低い演算子に対して，その演算子とオペランドの結果を仮のオペランドと置くことで，演算子が 1 つの式に変換する．優先度を知るために式を逆ポーランド記法に変形し，オペランドと演算子の結果を仮のオペランドと置いて，最終的に式が $E_1\ \Theta\ E_2$ ($\Theta \in \{<, <=, >, >=, =, <>\}$) の形になるまでこの動作を行う．例えば，$x + 2 * y > 0$ は，$AS_2 > 0$ ($AS_1 = 2 * y,\ AS_2 = x + AS_1$) と変形できる．

3.2.2. データ生成

まずは，$E_1\ \Theta\ E_2$ を $E\ \Theta\ V$ の形に変換することを考える．V は具体的な値である．E_1，E_2 が両方とも具体的な値でない場合は,E_2 に対してランダムな値を与え，$E_1\ \Theta\ E_2 \Leftrightarrow\ E_1 \Theta\ V, E_2 = V$ のように変形して，それぞれの条件に対してテストケースを生成する．E_2 が具体的な値の場合は，演算子と値によってテストケースを生成する．E_1 が具体的な値の場合は，V reverse(Θ) $E_2 \Leftrightarrow\ E_2\ \Theta\ V$ と変形する．reverse は，引数の反対の意味の演算子を取得する関数とする．

$E\ \Theta\ V$ の形に変換出来たら，演算子と V によって $E\ \Theta\ V$ を満たす値 V_E を求める．E が仮のオペランドであれば，$V_E = \Psi\ (a_1, a_2, ..., a_n)$ に対しても,$a_1, a_2, ..., a_n$ の値を求める．Ψ は演算子，$a_i (i \in \{1, ..., n\})$ は Ψ のオペランドである．a_i が仮のオペランドであれば同様の動作を行い，TC に含まれる入力変数 S_{iv} の値が求まるまで繰り返す．

4. 作成したツール

今回開発したツールの画面は次の 1 のようになった．主な機能は「エディタ」,「テストケース生成」,「評価」の 3 つに分けられる．エディタで，入力変数 S_{iv}，出力変数 S_{io}，FS(TC，DC) を入力する．章 3 で説明した方法を用いて，S_{iv} と TC からテストケースを求める．評価機能は，Defining-Condition に，テストケースと S_{ov} を代入して，真偽を判定する．S_{ov} は手動で入力する．

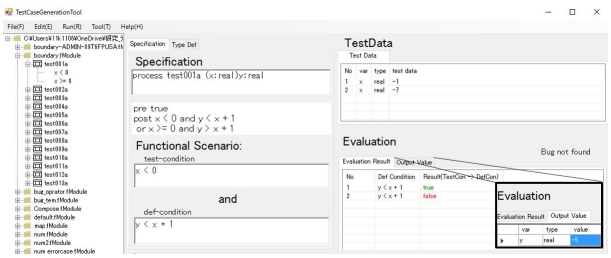

図 1: ツールの画面

5. 評価実験

テストケースの生成能力を比較するために，SMT-Solver と本ツールを比較する．SMT-Solver はテストケース自動生成によく利用される [4]．比較対象は，SMT-Solover の中でも代表的な Z3 と比較を行う．Microsoft 社の rise4fun(http://rise4fun.com/Z3) を利用して，Z3 の制約を記述してテストケース生成を試みる．

実験方法は，様々な演算子による共通の制約を用意して，本ツールと SMT-Solover で条件を満たす値を生成出来るかどうかを試みる．例えば，今回は，6 つの演算子，制約の数 31 つを実験対象として比較を行った．実験を行った結果，次の表 1 のようになった．

表 1: データ型毎のテストケース生成結果

演算子の型	用意した制約の数	Z3	My tool
数値型	6	6 (6/6)	3 (3/6)
列型	7	1 (1/6)	6 (7/6)
集合型	6	0 (0/6)	6 (6/6)
複合型	6	0 (0/6)	6 (6/6)
写像型	6	0 (0/6)	6 (6/6)
total	31	7(7/31)	27 (27/31)

数値型以外の演算子に対しては，Z3 よりも本ツールの方が生成能力が高かった．Z3 が生成出来たのは列型の演算子 head を使った制約に対してのみであった．数値型の制約に対しては，Z3 の方が優れていた．本ツールでは，制約に同じ変数があった場合は（$x * x < 5$ 等）失敗する場合が見られた．

6. 結論と今後の課題

形式仕様から自動的にテストケース生成する方法と評価するツールを開発し，テストケースを自動生成することを試みた．評価実験として，テストケースの生成能力を比較するために，SMT-Solver と本ツールのテストケース生成能力を比較した．実験結果から，写像や集合などの数値型以外の制約に対しては，本ツールは SMT-Solover よりもテストケース生成に成功した数が多いことがわかった．

参考文献

[1] Phill Stock and David Carrington, A Framework for Specification-Based Testing, IEEE TRANSACTION ON SOFTWARE ENGINEERING, VOL 22, NO.11, 1996.

[2] Saoying Liu, Formal Engineering for Industrial Software Development Using the SOFL Method Springer Germany. 2004.

[3] Saoying Liu, A Decompositional Approach to Automatic Test Case Generation Based on Formal Specifications, Secure Software Integration and Reliability Improvement, p147-155, 2010.

[4] Khalek, S.A. , Guowei Yang ,Lingming Zhang, Marinov, D. ,Khurshid, S. , “ TestEra: A Tool for Testing Java Programs using Alloy Specifications ”， IEEE Automated Software Engineering, pp. 608 - 614，2011 .