モデル検査の振舞駆動開発への応用

†小倉　真人　†深海　悟

†大阪工業大学　大学院情報科学研究科

1．はじめに

近年，システムの仕様の誤りを早期に発見するための方法として，モデル検査が注目を浴びている．

モデル検査では仕様の誤りが発見された場合，反例となる振舞を提示する．この反例を利用したテストケースを生成する方法が研究されている[1]．しかし，テストが後工程で行われるため，修正のために前工程への手戻りが発生する．また，テストが手作業で行われている場合では，テストの漏れや間違い起きることがある．

そのため，テスト駆動開発(TDD)[2]のように，テストを元に開発を行うスタイルやテストの自動化が注目を浴びている．なかでも，TDDの発展である振舞駆動開発(BDD)では，TDDよりテストコードの可読性が上がり、テストコードそのものが要求仕様となりうるとして注目されている．しかし，要求仕様からテストコードを生成するには，要求仕様の理解とシステムの振舞いを理解することに非常に多くの手間と時間がかかることが問題として挙げられる．

そこで本稿では，BDDに対応したテストコード(RSpec)の半自動生成と，BDDを支援する環境を提案する．提案の内容は次の2つである．

(1) モデル検査ツールNuSMV[3]が出力する反例を利用して，テストコードの出力を行う．
(2) テストコード，プログラムコードに変更があった際に自動的にテストを行い結果の通知を行う．

また，本稿では提案した環境の実装を行い，適用実験を行った結果についても述べる．

2．反例を利用したテストコードの生成

NuSMVを始めとする多くのモデル検査ツールは，モデルの誤りを発見した際に反例を出力する．反例とは誤りが発生するまでのモデルの振舞いである．本研究では，この反例を出力する機能を利用することでテストコードの生成を行う．そのため，要求仕様として正しい振舞いを，あえて反例として出力させる検証式を記述することで，反例の出力を行う(図1)．

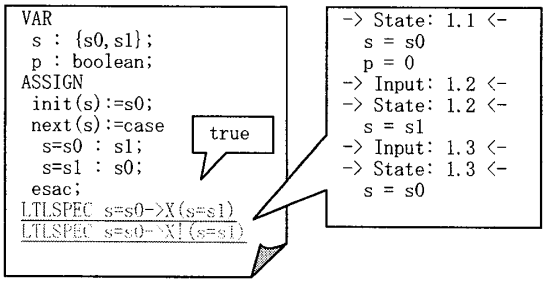

図 1 ： 反例の出力方法

また，一部の検証式に関しては，NuSMVソースコードを解析することで自動生成が可能である(図2)．

```
s : {s0,s1};
p : boolean;                          変数宣言を元に
SPEC AG p; SPEC AG !p                 検証式を生成
SPEC AG !s=s0; SPEC AG !s=s1
```

```
next(s):=case
 s = s1 & p : s1;                     変数の遷移を元に
 ...                                  検証式を生成
LTLSPEC G((s = s0 & p) -> X!(s = s1))
LTLSPEC G((s = s0 & !p) -> X(s = s1))
```

図 2 ： 検証式の自動生成例

出力として得られた反例ごとの振舞いを解析し，変換することでテストコードとして出力を行う．その際に，反例の中に存在するテストに必要ない変数の削除を行うことで，より理解のしやすいテストコードの作成が可能となる(図3)．

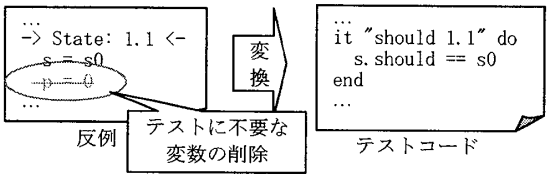

図 3 ： テストコードの生成例

3．テストの自動化と結果の通知

次に，テストコード，プログラムコードに変更があった際に自動テストを行い，結果を通知する方法について述べる．

BDDではテストコードを実行することで，作成

Application of model checking to behavior drive development
†Mahito Ogura, Satoru Fukami
Osaka Institute of Technology Graduate School of Information Science and Technology

したプログラムの動作が正しいかどうかを自動的にテストすることが出来る. しかし, BDD では以下の開発サイクルを繰り返して行うため, 開発規模が大きくなるとサイクルの回数が増え, 必然的にテストコードを実行する回数が増加する.

(1) 失敗するテストを書く(Red)
(2) テストがパスするような最小限のコード本体を書く(Green)
(3) リファクタリングを行う(Refactor)

そこで, 本提案では特定ディレクトリ下に存在するテストコード, プログラムコードに対して変更があった際に, 自動的にテストコードの実行を行い, 実行結果の通知を行う(図 4).

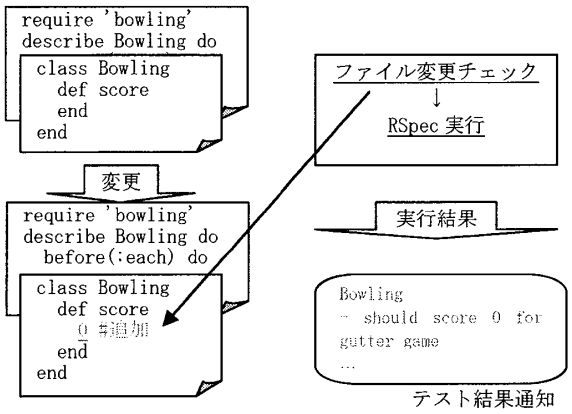

図 4 : 自動テストと結果通知の例

これにより, 変更が加えられるたびテストコードが実行されるため, ユーザがソースコードの変更のたび, 手動でテストを実行する必要がなくなる. また, テストの結果をユーザの目のつきやすい場所に表示することで, よりユーザにテストを意識させるといった利点がある.

4. 実装と適用実験

提案した 2 つの内容について実装および開発環境の構築を行った.

(1) 反例を利用したテストコード生成機能を Ruby で GUI アプリケーションとして実装した. 実現した機能は以下の通りである.
  - NuSMV ソースコード用編集エディタ
  - NuSMV を使ったモデル検査の実行
  - 変数の遷移に関する検証式の自動生成
  - 反例からテストに必要な変数を選択
  - 編集した反例をテストコードに出力
  - テストコード用編集エディタ

(2) 自動テストには, Ruby のテスト用ユーティリティ ZenTest[4]に含まれる autospec を利用した. また結果通知には Mac OS X 用のイベント通知アプリケーション Growl を autospec と連携させることで, テストの実行結果を通知する環境を構築した.

これらを利用した BDD が可能か調べるため, 適用実験として Rails を用いた blog の開発を以下の手順で行った.

(1) blog のモデル化を行い, 振舞いを NuSMV で検証し, NuSMV が出力した反例から不要な変数の削除を行い, テストコードを出力した.
(2) 出力されたテストコードを元に開発を行い, テストが Green となる実装.
(3) すべてのテストが Green になった時点でシステムが要求仕様を満たしているかの確認.

適用実験の結果は以下のとおりである.

- 反例から生成したテストコードを利用した, BDD を行うことができた.
- コードに変更が加えられるたび, テストが自動実行されることで, 開発の効率化が期待できる.
- テスト結果が目につきやすい位置に表示されることで, テスト結果を意識しやすくなった.
- 出力されたテストコードにおいて, 現在の Matcher よりも適切な Matcher が存在する場合は手作業での修正が必要.

5. おわりに

本稿では, モデル検査ツールが出力する反例を利用したテストコードの生成と, テストの自動化についての提案を行った. また, 提案した環境を実装し, 適用実験を行った結果についても述べた. 最後に, 今後の課題を以下で述べる.

- 従来手法と提案手法との間における開発効率についての比較調査.
- 要求仕様として正しい検証式から, 反例を出力させる検証式生成アルゴリズムの開発.
- NuSMV ソースコードや検証式からテストに必要な変数の自動抽出アルゴリズムの開発.
- 反例の編集時に Matcher の選択候補を提示, 選択することで, 適切な Matcher をテストコードへ出力する機能の開発.

参考文献

[1] Gordon Fraser. Automated Software Testing with Model Checkers. Dissertation, Graz University of Technology, 2007.
[2] Kent Beck, Test-Driven Development: By Example, The Addison-Wesley Signature Series, 2003
[3] NuSMV, http://nusmv.irst.itc.it/
[4] ZenTest: Automated test scaffolding for Ruby ,http://www.zenspider.com/ZSS/Products/ZenTest/
[5] Growl, http://growl.info/