モデル検査を利用した仕様とソースコード間におけるシステムの振舞いの不一致発見

青木善貴[†1][†2]　　松浦佐江子[†2]

開発現場では不十分な要件定義や実装時のミス・誤解等により人手では発見困難な不具合が発生する．担当する要員により原因特定にかかる工数が著しく違うことは，開発プロジェクトの進捗の妨げになるため不具合原因を安定的に特定できる手法が必要である．本研究ではモデル検査の網羅的に要求される性質が満たされるかを自動検査する特性を利用して Java ソースコードに対して，仕様とソースコード間の振舞いの不一致の原因の特定を行う．

Behavioral Inconsistency Detection between Source Code and Specification using Model Checking

YOSHITAKA AOKI[†1][†2]　SAEKO MATSUURA[†2]

Software programs often include many defects that are not easy to detect because of the developers' mistakes, misunderstandings caused by the inadequate definition of requirements, and the complexity of the implementation. Due to the different skill levels of the testers, the significant increase in testing person-hours interferes with the progress of development projects. Therefore, it is for any non-specialized developer to identify the cause of the defects. Model checking has been favored as a technique to improve the reliability earlier in the software development process. In this paper, we propose a verification method in which a Java source code control sequence is converted into finite automata in order to detect the cause of defects by using the model-checking, which has an exhaustive checking mechanism.

1. はじめに

開発現場では不十分な要件定義や実装時のミス・誤解等により人手では試すべき想定ケースが多すぎて網羅しきれず原因の特定が困難な不具合が発生する．モデル検査は状態遷移系として定義されたシステムに対し，要求する性質の論理式が満されるかを検査する手法である．

本研究の目的は，モデル検査を用いてシステムの振る舞いの仕様と実際のシステムの振舞いの不一致の発見を開発現場の開発者でも可能にすることである．提案手法では検査モデル作成の準備として，仕様を分析してモデル化対象になる文言を抽出し，それを整理してデシジョンテーブルにまとめる．本論文ではその分析・整理のロジック及び提案する検査手法の有効性について議論する．

2. ソースコードの検証手法

2.1 手法の概要

我々はモデル検査技術を用いてシステムの振舞いの仕様から作成した検査モデル(仕様モデル)とソースコードの制御フローに基づき作成した検査モデル（ソースコードモデル）を結合して検査することで，仕様とソースコード間のシステムの振舞いの不一致を発見する手法を提案してきた[1][2]．本研究では業務システムを念頭に置き，ユースケースレベルのシステムの振舞いを検査の対象としている．システムの振舞いはシステム内の処理とユーザの手作業の組み合わせで表わされる動作の連なりである．従って，システムの振舞いはユースケースを構成するシステムの基本機能を表すアクションと基本フロー，事前・事後・分岐条件により表せる．これらを抽出し(図 1①)，デシジョンテーブルで整理する(図 1②)．デシジョンテーブルは機能仕様をアクション，事前・分岐条件から整理するものである．本研究ではさらに実行結果(事後条件)も記述し，システムの振舞いを表せるようにした．このシステムの振舞いを状態遷移に置き換えて仕様モデルを作成する(図 1③)．次に仕様モデルが想定外の状態遷移を起こさないことを検証する検査式を作成する(図 1⑤)．

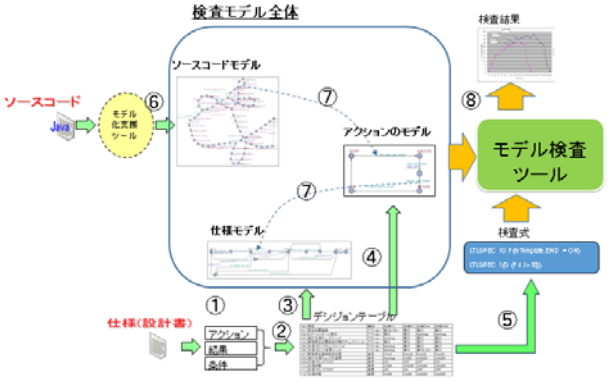

図 1　仕様モデルによる検査

†1 日本ユニシス株式会社
Nihon Unisys ,Ltd.
†2 芝浦工業大学
Shibaura Institute of Technology

ソースコードモデルはユースケースレベルの機能に限定したソースコードの階層構造に基づいたモデル変換により作成する(図 1⑥). モデル化されたステートメントは検査モデル上で状態を表すロケーションと紐付く．ユースケースレベルの機能(アクション)はメソッドに対応すると仮定できるので，メソッドをソースコードと仕様の接点として取らえて，これを機能の中継のみを目的とするアクションモデルとして定義し(図１④)，ソースコードモデルのメソッド呼び出し部と仕様モデルの属性の状態変化部をアクションモデルで紐付けして検査モデルを構成する(図 1⑦). この検査モデルを検査式により検査する(図 1⑧). 検査は以下の手順で行う．図１⑦で構成された検査モデルはユースケースレベルの機能を表したものになる．手順１はこの検査モデルがソースコードの制御フローの動きを再現できていることを到達可能性の検査で確認する．手順２は作成した検査式による検査でシステムの振舞いに合致しない状態遷移が起きないことを確認する．ただしアクションモデルの基のメソッドのロジックに問題がある場合，これまでの検査では不具合は発見できない．その場合は，アクションモデルをソースコードモデルへ再変換して，新たなソースコードモデルを追加した検査モデルで上記手順を繰り返す．そうすればモデルが段階的に詳細化され，該当メソッドのロジックについても検査できる．本研究はユースケースレベルのシステムの振舞いの仕様をデシジョンテーブルで整理してモデル化し，想定外の振舞いがないことを検証している．従って振舞いの仕様以外については対象外となる．

2.2 現状の問題点

日本語は表現の自由度が高いためモデル化に際し，仕様の分析・整理が難しい．この分析・整理のロジックが明確でなかったため今回このロジックの明確化に取り組んだ．

最初に文章の単文及び行為文への変換を行う．関係性が不明確な存在文(・・ある)を，関係性が明確になる行為文(・・する)に変換する．次に動作動詞・可算名詞の抽出を行う．英文における動作動詞がメソッドに該当し，可算名詞がクラスに該当するとする研究がある[3]．この考え方に基づき可算名詞と動作動詞を含む文章を抽出し，これらが含まれる文章を基にデシジョンテーブルを作成する．以降は既存の提案と同様の手順で仕様モデルを作成する．

また以前の適用事例はアクションモデルからソースコードモデルへの再変換による詳細化までは行っていなかった．今回は実際に作成されたプログラムを適用事例とし，アクションモデルからソースコードモデルへの再変換による詳細化まで含めた提案手法の有効性の確認を行う．

3. 適用事例

3.1 適用事例概要

長方形エディタプログラムへ提案手法を適用した．これは芝浦工業大学のプログラミング演習の課題である．提示された仕様書と作成された Java プログラムへ適用した．プログラムの機能は長方形の作成・削除・移動・拡大・縮小である．

3.2 自然言語の仕様からの仕様モデルの作成

既存の手法に今回の提案を加え検査モデルの作成を行った．結果として長方形の個数・はみ出し状態・重複の状態がモデル化された．またデシジョンテーブルよりアクション＋条件＋結果の組み合わせがわかるため，検査すべきシステムの状態は以下の 3 点となった．

- 長方形がボードからはみ出している
- 長方形の数がマイナスまたは 11 以上
- 同じ長方形が存在する

3.3 モデル検査の実施

手順１の到達可能性の検査を行ったが不具合は発見されなかった．手順２の検査として，先にあげた 3 つの「想定されない状態」について検査を行った．拡大処理で長方形が重複する不具合が発見された．反例から読み取れる不具合の発生条件を念頭におきソースコードの該当部分を調査することにより拡大処理において重複チェックがなされていないという不具合の原因が特定できた．

今回の事例ではプログラム側で，はみ出しの不具合が確認されていたので，はみ出しチェックのメソッドのアクションモデルをソースコードモデルに変換して検査を行った．結果としてはみ出しの発生が検出された．反例より確認すると長方形の基点の X 座標=0，Y 座標=0 の状態で作成処理を行うとはみ出しとなることがわかった．これは誤判定であり，ソースコードを調査すると条件判定式の図 2 の○印の部分に等号が不足していることが原因と判明した．

```
if(x >= 0 && y >= 0 && x + w <= board_w && y + h <= board_h){
```

図 2　条件判定式

4. まとめ、今後の展望

自然言語で実際に記述された仕様を提案手法で分析・整理することでモデル化できた．また学生が作ったソースコードを検査対象としたことにより，実際の適用場面における提案手法の有効性も確認できた．今後の課題はより業務寄りのプログラムへの適用による有効性の確認である．

参考文献

1) 青木善貴，松浦佐江子，モデル検査技術の開発現場への適用，電子情報通信学会技術研究報告 KBSE , 113(160)，pp.91-96，2013
2) Aoki, Y., Matsuura, S., "Verifying Business Rules Using Model-Checking Techniques for Non-specialist in Model-Checking", IEICE TRANSACTIONS on Information and Systems Volume E97-D No.5, pp.1097-1108 2014.
3) 金田重郎，井田明男，酒井孝真，英語 7 文型と関数従属性に基づくクラス図の理解，電子情報通信学会技術研究報告 KBSE, 113(475)，pp.31-36，2014