モデル検査に基づくプログラム欠陥抽出作業支援ツールの開発と実践

青木　善貴†　　松浦　佐江子†
芝浦工業大学　大学院　工学研究科　電気電子情報工学専攻†

1. はじめに

システム開発の現場において，開発者は必ずしも提示された仕様から「設計者の意図したとおりのプログラムを作成できる理解」を持てるとは限らない．

開発者は設計者との業務知識等の齟齬により，設計者の正確な意図に気付かずに仕様を理解する．そのような理解に基づいて作成されたプログラムは，一見正しく動作するが不具合を内在していることが多い．この内在した不具合は発生条件が見つけにくく，なかなか再現できない．これを解消するためには，緻密な調査と徹底したテストが必要になり，多大な工数が発生する．

われわれは，こうした不具合をソースコードから生成されるプログラムの状態遷移モデルに対して，モデル検査ツールを用いて発見する手法を提案してきた[1][2].

この手法において検査者は，モデル化範囲を設定して検査モデルの作成及びモデル検査を行い，その結果を解釈して知見を導き出す．これを人手で行うとモデル検査知識が必須となるため，検査できる要員が限定されて幅広く利用してもらうことができない．

そのため，プログラムの基本仕様さえわかっていれば，経験の浅い検査者でも不具合を発見できるように検査支援ツールを用意することとした．このツールは「モデル化範囲の設定」から「検査モデルの自動生成」「容易な検査」までをサポートする．本稿では，このツールの開発から実践までを提示し，その有効性を議論する．

2. モデル検査に基づく欠陥抽出手法概要

稀有な条件で発生する不具合をモデル検査手法を用いて発見する手法である．

検査者は基本仕様を理解しており，不具合の現象を「ある機能に関る何らかの操作で無限ループが発生する，テーブルロックが発生する」といったレベルで捉えているとする．こうした現象を容易に定義したり，検査条件を容易に設定できるようにする．

以下に機能的特徴を挙げる．

(1)モデル検査ツールの網羅的チェックを用いた不具合原因の特定
(2)モデル検査ツールの反例・シミュレーション機能を用いた高い確度での不具合再現
(3)修正モデルの再検査による不具合解消の確認
(4)検査支援ツールの以下の機能による検査の容易さ
- モデル化範囲の設定
- 検査モデルの自動生成
- 容易な検査

Development and Practice of Program Defect Detecting Work Supporting Tool Based on Model Checking
†Aoki Yoshitaka　†Matsuura Saeko
†Graduate School of Engineering, Shibaura Institute of Technology Course of Electronic Engineering and Computer science

3. 検査支援ツールによるモデル検査手順

図1は検査支援ツール(図1)の起動画面である．モデル化範囲の設定では，Excelをインターフェースとし，支援機能の処理をVBAで実装する．Javaソースの XML 変換には JJMLT[3]を用い，自動生成される検査モデルのモデル検査ツールは UPPAAL[4]を使用する．自動生成で用いるモデル定義の格納にはAccessを使用する．

本提案手法では，以下の手順で検査を行う．

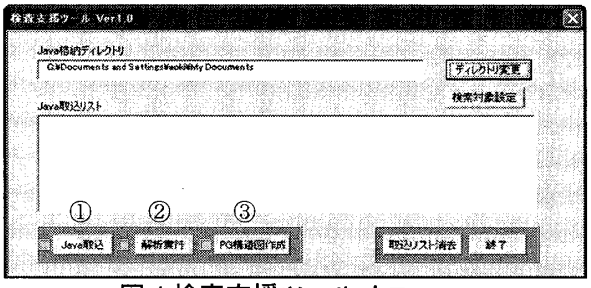
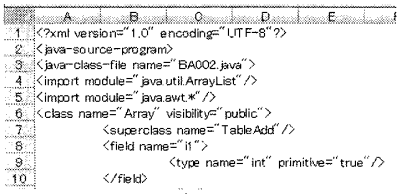


図1検査支援ツールメニュー

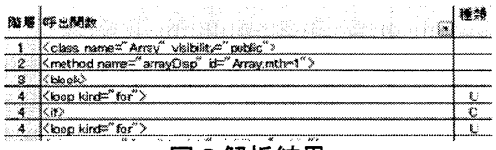
手順1：Javaソースコードより XML 作成(図 1-①実行).
ツールより JJMLT を起動して Java コードを XML 形式に変換して Excel に取り込む(.図 2).



```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <java-source-program>
3  <java-class-file name="BA002.java">
4  <import module="java.util.ArrayList" />
5  <import module="java.awt.*" />
6  <class name="Array" visibility="public">
7          <superclass name="TableAdd" />
8          <field name="i1">
9                  <type name="int" primitive="true" />
10         </field>
```

図2XMLデータ取込結果

手順2：着目する構成要素の洗い出し(図 1-②実行).
構成要素のつながりを XML データより抽出する(図3). 例では無限ループが生じる．条件を検査するために"for"と"if"に着目している．



階層	呼出関数	種類
1	<class name="Array" visibility="public">	
2	<method name="arrayDisp" id="Array.mth=1">	
3	<block>	
4	<loop kind="for">	L
4	<if>	C
4	<loop kind="for">	L

図3解析結果

手順3：プログラム(PG)構造図の作成(図 1-③実行)
手順2で作成したデータを基に構成要素に依存するプログラムの構造図を作成する(図4).

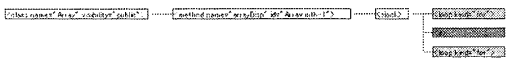
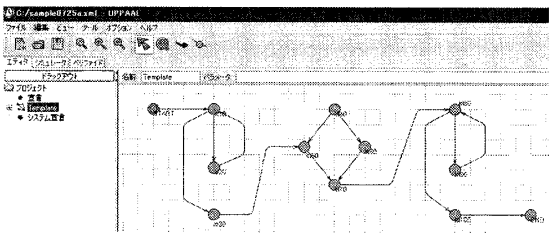


図4プログラム構造図

手順4：モデル検査ツール用ファイル作成
PG構造図より検査部分を選択し，ダブルクリックモデルを生成しモデル検査ツールの形式で保存．

手順５：モデル検査ツールによる検査
手順４で作成されたファイルを読込み，モデル検査ツールを起動・検査を行う．



4. 基本モデル生成

モデルを自動生成するために，構成要素となる基本モデルは予め Access 上にその構造を定義しておく．モデル作成時に検査支援ツールが該当する定義を読み取りモデルを作成し，複数ある場合は組み合わせる．基本モデルの定義の例は以下のようになる(図７)．

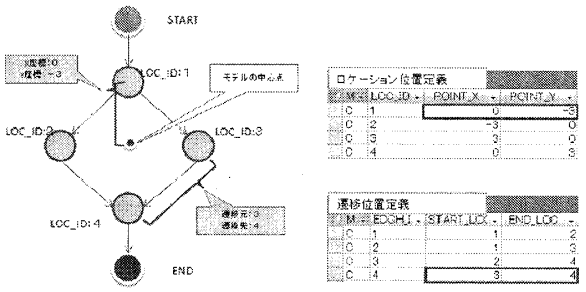


図７基本モデル(if)定義概要

5. プログラム状態の抽象化

本手法では，既存のソースコードをリバースしてモデルを生成するため，プログラムをそのままモデル化すると状態爆発を起す可能性が非常に高い．それを防ぐためには，プログラムを抽象化して検査したい状態のみをモデルにする必要がある．

例として無限ループの検査を示す．画面インターフェースを有し，入力データに処理結果を返す機能を持つプログラムを想定した状態遷移を以下に示す(図８)．

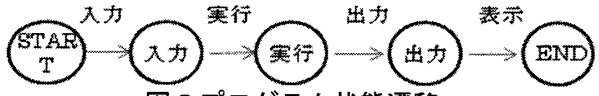


図８プログラム状態遷移

無限ループが発生したとすると，検査したい現象はループに関する状態だけであるから状態遷移は次のようになる(図９)．

図９ループに着目した状態遷移

図９が検査モデルの大枠となる．抽象化の手順として，まず検査対象となる現象の特徴的な状態を抽出する．この抽出した状態を非決定的な動作として捉えて，原因追求の視点で段階的に詳細化する．それより下位の状態は考慮しないことにより抽象化を行う．無限ループの場合の特徴的な状態は”ループ”であり，これを段階的に詳細化すると非決定的な動作は Level1～5 のように捉えられる．(さらなる細分化も可能であるがここでは議論しない)

Level1：ループ/非ループを非決定
Level2：ループ条件(True/False)を非決定
Level3：ループ条件操作処理(実行/未実行)を非決定
Level4：ループ条件操作処理の実行条件(条件値)を非決定
Level5：ループ条件操作処理の実行条件(特定範囲内の条件値)を非決定

Level1～4 は実行部のみモデル化，Level5 は実行部・入力部をモデル化する．Level が上がるほどモデルの複雑さは増す(図１１)．この無限ループの例は発生状況より”for””if”が関係することが想定されるため Level4 からモデル化を行った．Level4 でも原因の発見はできるが発生しえないデータが含まれる可能性がある．Level5 では，入力部のモデル化も行うため実運用で発生しうるデータであるかの確認まででできる．(詳細は[1][2]参照)

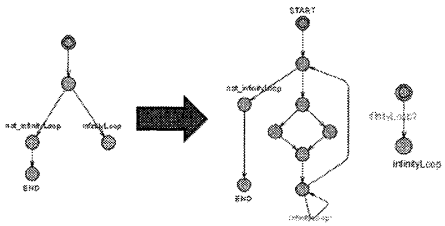


図１１Level1 から Level4 へのモデル変化

抽象化の問題として，抽象化し過ぎてしまうと現象を確認できなくなることがある．そのような場合は抽象化を調整して再度検査を行う必要がある．

モデルの自動生成を行う場合，この調整は個人の知識・経験で行うのではなく，検査者が現象の発生状況及び検査状況を鑑みて，予め決められた手順に従い段階的に行えるようにすべきである．それには前述した抽象化の Level の考えを用いることが適当と考える．

6. まとめと課題

検査支援ツールは現在開発中であるが，「基本モデル生成」の基本部分は実装済みであるので，Java ソースコードから簡単なモデル構造の生成は可能である．これを基に条件式を手動で設定して検証を行い欠陥の抽出ができたことにより有効性が確認できた．

今後はより複雑なモデルに対応できるようにし，「抽象化」については今回示した抽象化 Level をツールの実装に反映させる予定である．

課題は適切な抽象化 Level を選択する方法や，生成モデルへ条件式を設定するためのプログラムの Level 合った抽象化方法の確立である．

参考文献

[1] 青木善貴，松浦佐江子，ソースコードの解析を利用したモデル検査に基づく欠陥抽出手法の提案，第８回情報科学技術フォーラム,pp.387-391, 2009
[2] 青木 善貴，松浦佐江子，ソースコード解析を利用したモデル検査に基づく欠陥抽出手法の提案，知能ソフトウェア工学研究会, KBSE2009-44,pp.79-84, 2009
[3] JJMLT:HTTP://WWW.HPC.CS.EHIME-U.AC.JP/~AMAN/PROJECT/JJMLT
[4] UPPAAL:HTTP://WWW.UPPAAL.COM/