Vol.2015-SE-187 No.40
2015/3/13

モデル検査における不具合原因特定手法

鷲見毅[†1]　和田大輝[†1]　晏リョウ[†1]　武山文信[†1]

近年，ソフトウェアの大規模化に伴い，開発の下流工程におけるテストだけではソフトウェアの品質確保が困難になっている．その為，開発の上流工程における品質確保の必要性が高まっており，その手段としてモデル検査が注目されている．しかし，モデル検査を開発で実践適用するためには幾つかの課題がある．そのひとつとして，発見した不具合の原因箇所の特定が困難ということが挙げられる．本稿では，モデル検査によって発見された不具合に対して，その原因箇所を自動特定する手法を提案する．

Method for identifying causative conditions from counter example

TAKESHI SUMI[†1]　TAIKI WADA[†1]　RYO AN[†1]　FUMINOBU TAKEYAMA[†1]

Recently, as software becomes larger-scaled, it becomes difficult to ensure its quality only by testing in the end of the development. Therefore, there is an increasing need to ensure the quality in the development of the upstream processes. And as a means, model checking has been attracting attention. However, to take advantage of the model checking in practical, there are several challenges. One of them, it is that is difficult to determine the cause of the defects. In this paper, for the defects that was discovered by model checking, we propose a method to automatically identify the cause place.

1. はじめに

近年のソフトウェアの大規模化により，テストだけではソフトウェアの品質を確保することが困難になっている．その為，上流工程において設計書等の開発成果物の検査を行い，早期に不具合を発見することが求められている．こうした上流工程における品質確保の手段のひとつにモデル検査が挙げられる．モデル検査は，検査対象の振る舞いを状態遷移モデルとして記述した検査用モデルと，検査対象が満たすべき「正当でない状態に決して陥らない」，「特定の状態に必ず到達できる」等の性質を表す検査式を入力として，検査モデルを網羅的に探索することで，検査式に対する違反の有無を明らかにできる．

しかし，実際のソフトウェア開発でモデル検査を実践的に用いる為には，幾つか解決すべき課題がある．そのひとつとして，反例を解析する作業の困難さが挙げられる．モデル検査において検査式に対する違反，即ち不具合が発見されると，発見された不具合に至るまでの実行履歴が反例として出力される．不具合を修正する為には，出力された反例を人手で解析して不具合の原因箇所を特定する必要があるが，反例の解析はノウハウと時間を必要とする困難な作業になっている．

本稿では，こうした原因箇所の特定を自動で行う手法を提案する．また，幾つかの事例に対する提案手法の試行によって，不具合の原因箇所を正しく特定でき，提案手法が有効であるかを評価する．

本稿は，2 章でモデル検査について述べ，3 章でソフトウェア開発に適用する上での課題について説明する．その上で，4 章で提案する原因特定手法について詳細な説明を行い，5 章でその適用と評価について述べる．6 章で関連研究を紹介し，最後に 7 章でまとめと今後の課題を述べる．

2. モデル検査

2.1 モデル検査の概要

モデル検査は，検査モデルに含まれる全ての状態遷移を網羅的に探索する事で，検査対象が満たすべき性質に違反するかどうかを検査する．その為，テストでは再現が困難な実行タイミングや複雑な条件の組み合せによって発生する不具合の発見に効果が期待できる．モデル検査の実施は，図 1 で示される流れで行われる．

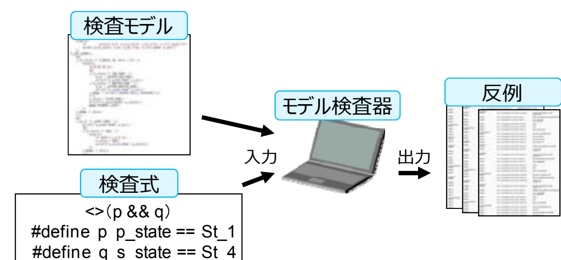


図 1 モデル検査実施の流れ

図 1 に示す流れにおいて，モデル検査を実施する為の作業として大きく次の 3 つの作業が必要にある．

1) モデル検査の入力の作成

入力の作成では，検査対象を状態遷移モデルとして記述した検査モデルと，検査対象が満たすべき性質を記述した検査式を作成する．検査モデルは，仕様書や設計書から検査に必要な情報を抽象化して作成する．検査式も同様に仕様書や設計書から満たすべき性質を抽出し，論理式として記述する．

†1 (株)東芝 ソフトウェア技術センター
TOSHIBA Corporation Software Engineering Center

ⓒ 2015 Information Processing Society of Japan

2) 検査の実施

モデル検査において検査そのものは，モデル検査器によって機械的に行われる．その為，モデル検査の実施者は入力を与え，モデル検査ツールを実行するのみで良い．

3) 出力結果の解析

モデル検査器は，検査式に対する違反，即ち不具合を発見すると，どの様な状態遷移によって違反に至ったかを記録した反例を出力する．この反例を解析して検査モデルの問題点を発見し，これを修正し，再度モデル検査を行う．これによって，検査対象が正しい振る舞いをする様にしていく．

2.2 モデル検査ツール SPIN

本稿で扱うモデル検査ツール SPIN[5]では，検査モデルを専用記述言語である Promela（Protocol & Process Meta Language）で記述する．また，検査式は LTL（Linear Temporal Logic）で記述をする．SPIN は，これらの入力から Promela で記述された検査モデルを網羅的に探索して不具合を発見できるモデル検査器を生成する．SPIN が生成するモデル検査器は，C 言語のソースコードとして出力される．実際のモデル検査は，このソースコードをコンパイルしたモデル検査器を実行する事で行われる（図 2）．

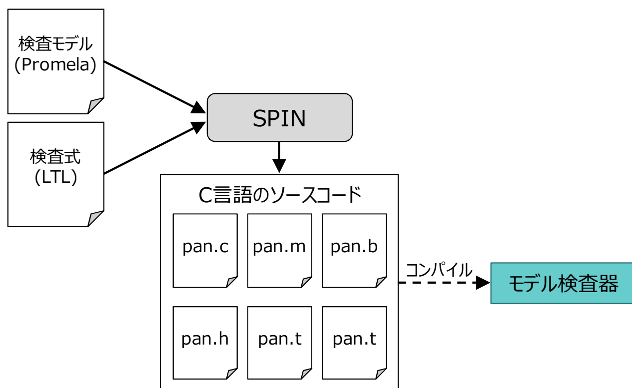


図 2 SPIN によるモデル検査器生成

3. モデル検査の適用上の課題

3.1 適用の課題と対策

モデル検査を実際のソフトウェア開発に適用する為には，幾つかの課題がある．我々は次の 3 つの課題を解決する事が，モデル検査をソフトウェア開発に適用する上で特に重要と考え，技術開発を行ってきた．

第 1 の課題は，モデル検査の入力の作成である．先に述べた様に，モデル検査ツール SPIN では検査モデルは専用記述言語である Promela で記述する必要があり，検査式は LTL で記述する必要がある．これらの記述方式は，一般的なソフトウェア開発者にとっては馴染みが薄く，作成が困難である場合が多い．そこで，検査用モデルの作成を自動化する為に，設計書等で用いられる状態遷移表を Promela で記述された検査モデルに変換する手法を開発した．もうひとつの入力である検査式についても，限定した日本語の組み合わせから LTL に変換する手法を開発した[1]．

第 2 の課題は，検査対象の大規模化によって探索すべき状態遷移の数が爆発的に増加する状態爆発と呼ばれる現象である．状態爆発が発生すると計算機の資源不足によって検査を完了する事ができない．これに対して，検査対象の状態を縮約して状態遷移の数を適切に削減する手法を開発した[2][3]．

第 3 の課題は，モデル検査の出力結果の解析である．モデル検査器から出力される反例は，不具合に至るまでの状態遷移の履歴が記述されるが，必ずしも可読性が高いとは言えない．そこで，反例の内容をチャート形式に加工し，可読性を向上させた反例チャートを出力する機能を開発した[1]. 反例チャートでは，検査モデルに含まれる変数の値，状態，発生したイベントについて，その変化を実行ステップに対応させて表示する（図 3）．

図 3 反例チャート

これらの手法を実装したモデル検査自動化ツールは，状態遷移表と日本語の組み合せから自動生成した検査モデルと検査式を用いてモデル検査を実行し，反例チャートを出力する．更に，反例チャートの各ステップと状態遷移表を対応させて表示させる機能も実装している（図 4）．

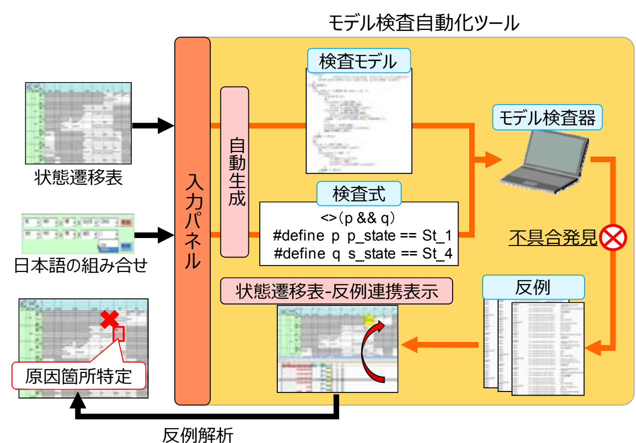


図 4 モデル検査自動化ツール

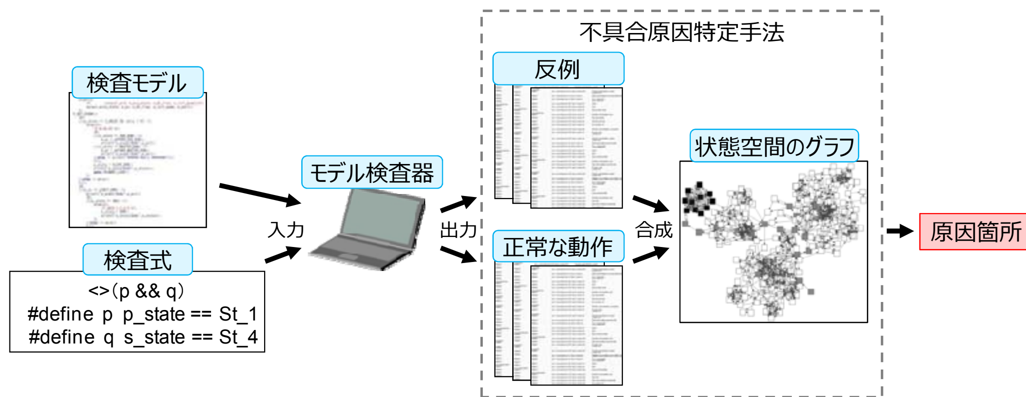


図 5 提案手法の概要

3.2 反例解析の課題

反例の解析を行う分析者は，反例チャートに記録された実行履歴と想定される正常な動作を比較し，正常な動作との差異から不具合の原因箇所を特定できる．3.1 節で述べたモデル検査自動化ツールは，反例チャートと状態遷移表を対応付ける機能によって正常な動作との比較を支援する事ができる．しかし，実行履歴から検査対象の振る舞いを解釈する作業は分析者が行う必要があり，モデル検査自動化ツールの機能では支援できない．また，検査対象の正常な動作を把握していなければ反例チャートに含まれる正常な動作との差異に気付く事ができない為，原因箇所を特定できる分析者が限られてしまう．

こうした事から，反例の解析を自動化し，原因箇所を特定する手法が必要と考えられる．

4. 不具合原因特定手法

4.1 提案手法のアプローチ

従来の人手による反例解析では，分析者が把握している正常な動作と，反例に記述されている状態遷移を比較し，その差異から原因箇所を特定している．そこで，これを自動化する為に，反例以外の実行履歴を正常な動作として用いる事とした．SPIN によるモデル検査では，検査モデルの状態遷移を網羅的に探索して不具合を発見した場合に，不具合に至るまでの状態遷移の実行履歴を反例として出力する．この時，検査の過程では検査式に違反しない状態遷移も探索されている．この検査式に違反しない実行履歴を正常な動作とすることで，機械的に原因箇所を特定可能になると考えた．

そこで，以下の手順で不具合の原因箇所を特定する手法を提案する．提案手法の全体像は，図 5 の様になる．

1) モデル検査の結果として，全ての反例と正常な動作を表す実行履歴を出力する
2) 出力された実行履歴から探索された状態空間をグラフ化する
3) グラフ化された状態空間から不具合になる条件が確定した時点を特定し，その時点の変数の値，状態，発生したイベントの組を原因とする

以降の節で，各手順の内容について，詳細に説明する．

4.2 実行履歴の出力

SPIN では，先に述べた様に検査モデルを探索するモデル検査器を生成して検査を行う．このモデル検査器は C 言語のソースコードとして出力される．従って，このソースコードに変更を加える事でモデル検査器の処理を変更する事ができる．検査時に全ての反例と全ての正常な動作を出力させる為に，以下の処理を追加する変更を行う．

- 検査式に違反しない場合の実行履歴を正常な動作として出力する
- 実行履歴に情報を追加する

通常，生成されたモデル検査器は，探索した状態遷移が検査式に違反している場合に，その状態遷移までの実行履歴を反例として出力し，検査を終了する．逆に探索した状態遷移が検査式に違反しない場合は，反例ではないとして次の状態遷移の探索を継続する．そこで，検査式に違反しない場合にも実行履歴を出力する条件分岐と処理を追加する事で正常な動作の実行履歴を出力させる（図 6）.

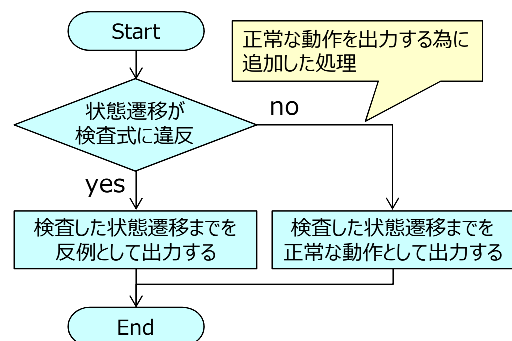


図 6 正常な動作を出力させる為の処理の追加

更に，出力される実行履歴に各変数の値，状態を表す ID，イベントを表す ID を出力する処理を追加する（図 7）．通常，モデル検査器が出力する反例には，プロセス ID と探索された状態遷移に対応する検査モデル上の位置が出力される．これに各変数の値，状態を表す ID，イベントを表す

ID を追加する事で，図 3 に示した反例チャートと同様の詳細度で実行履歴の内容を表す事ができる様になる．

状態を表すID
変数の値
イベントを表すID

```
 1:0:574:0,0,0:7:0
 2:1:556:0,0,0:7:0
 3:1:568:0,0,0:7:0
 4:0:574:0,0,0:7:0
 5:2:  0:0,0,0:7:24
 6:0:574:0,0,0:7:24
 7:2:310:0,0,0:7:24
 8:0:574:0,0,0:7:24
 9:2:320:0,0,1:4:24
10:0:574:0,0,1:4:24
11:2:  1:0,0,1:4:24
12:0:574:0,0,1:4:24
13:2:  2:0,0,1:4:23
14:0:574:0,0,1:4:23
15:2:364:0,0,1:4:23
```

SPINが通常出力する情報

図 7 実行履歴の内容

情報を追加された実行履歴の 1 行は，記述された変数の値の時に，ID で表される状態において，ID で表されたイベントが発生した事を意味する．

また，全ての反例と正常な動作を出力させる為に SPIN のオプションを用いる．SPIN で生成されたモデル検査器を実行する際，オプション「-e」を使用する事で全ての反例を出力するまで探索を継続させる事ができる．前述の変更を加えたモデル検査器をオプション「-e」を使用して実行する事で 1 度の検査で全ての反例と正常な動作を出力させる事ができる．

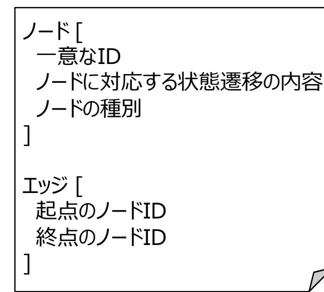
4.3 状態空間のグラフ化

モデル検査器によって探索された全ての状態遷移をまとめる事で，検査対象にどの様な状態遷移が含まれるかを明確にする．これによって，反例と正常な動作の差異を明らかにする事ができる．

モデル検査器の出力として得た全ての事項履歴の内容を，以下の基準でひとつのグラフにまとめる．

- 変数の値，状態を表す ID，イベントを表す ID が全て一致する場合は同一の状態遷移とみなす
- 正常な動作から得られた状態遷移と，反例から得られた状態遷移を区別可能にする ID を付加する

状態空間を表すグラフは，図 8 に示すデータ構造を持つノードとエッジで表す事ができる．1 つの状態遷移を 1 つのノードに対応させ，変数の値，状態を表す ID とイベントを表す ID の組で表す．ノードの種別は，「正常な動作に含まれる状態遷移」と「反例に含まれる状態遷移」，「検査式に違反すると判定された状態遷移（通常は，反例の末尾の状態遷移）」の 3 種類に区別する為の値を表す．状態遷移の実行順序はエッジによって表す．エッジの起点側の状態遷移が実行された後に，終点側の状態遷移が実行される事を意味する．


ノード [
  一意なID
  ノードに対応する状態遷移の内容
  ノードの種別
]

エッジ [
  起点のノードID
  終点のノードID
]

図 8 グラフのデータ構造

状態空間のグラフ化は正常な動作から行い，全ての正常な動作を表すグラフを作成する．この段階では，全てのノードの種別は「正常な動作に含まれる状態遷移」を表す値になる．作成されるグラフを図示すると図 9 の様になる．本稿では，説明の為にノードの種別に色を用いる．「正常な動作に含まれる状態遷移」は白，「反例に含まれる状態遷移」は灰色，「検査式に違反すると判定された状態遷移」は黒で表している．

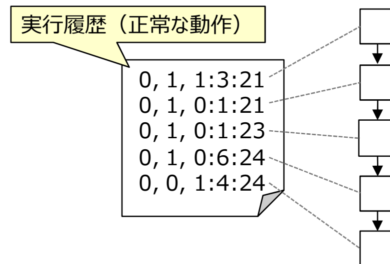


図 9 正常な動作から作成されるグラフ

正常な動作を全て読みこんだ後，反例を読み込む．読み込んだ反例に既出の状態遷移が含まれる場合は，ノードの種別を「反例に含まれる状態遷移」に更新する．既出の状態遷移ではない場合は，新規の状態遷移としてグラフ上の分岐として追加する（図 10）．

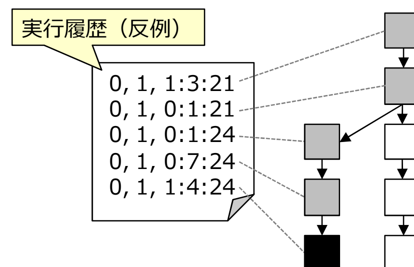


図 10 反例の実行履歴を追加したグラフ

この様にして，全ての反例と全ての正常な動作からモデル検査器によって探索された全ての状態遷移を 1 つにまとめ，状態空間全体を表すグラフを作成する（図 11）．

4.4 原因箇所の特定

モデル検査によって発見される不具合は，必ずしも直前の状態遷移のみが原因ではない．また，全ての状態遷移が

ⓒ 2015 Information Processing Society of Japan

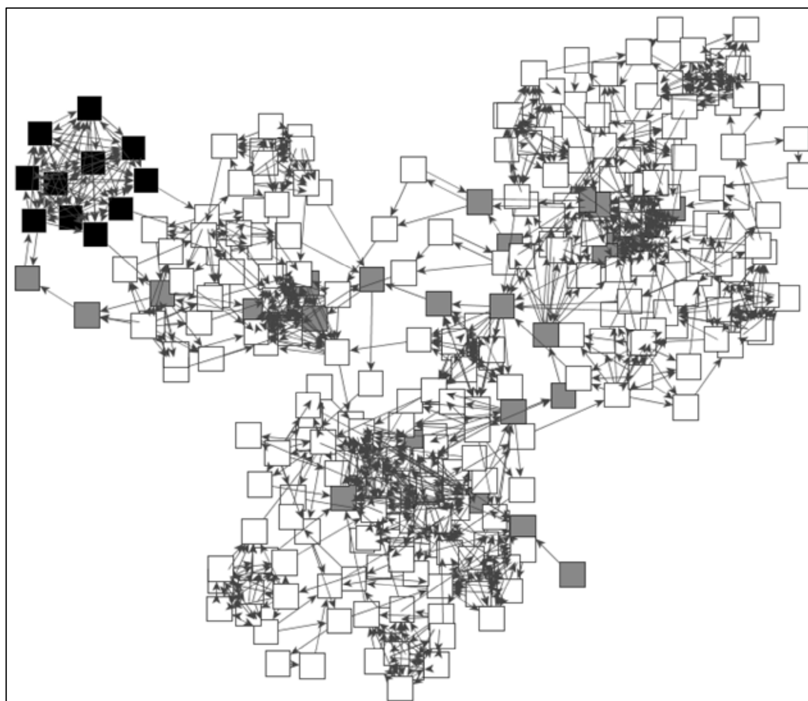


図 11 状態空間のグラフの例

不具合の原因である事はない．状態遷移のある時点において不具合になる条件が満たされ，そこから正常な動作に復帰できなくなると不具合になる事が確定する．従って，不具合になる条件が確定した時点の状態遷移が不具合の原因箇所と考える事ができる．

提案手法では，作成されたグラフから以下の手順で原因箇所を特定する．まず，モデル検査器によって出力された反例毎に，「検査式に違反すると判定された状態遷移」を起点として，反例の内容を逆順に辿ってグラフの探索し，最初に到達する「正常な動作に含まれる状態遷移」との分岐点を発見する．発見した分岐点に対して，起点とした状態遷移側に 1 つ進んだノードに対応する状態遷移を不具合の原因箇所として特定する（図 12）．

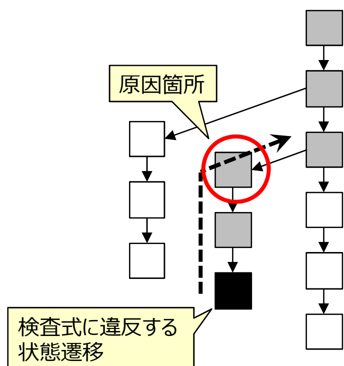


図 12 提案手法で特定される原因箇所

この手順で特定される原因箇所は，以下の特徴を持つ．

- 反例に固有の状態遷移で，正常な動作には登場しない
- その状態遷移が発生すると，必ず不具合に至る
- 不具合に至る場合には，必ずその状態遷移が発生する

これは，不具合になる必要十分条件を満たす状態を原因箇所として特定している事になる．原因箇所に対応する状態遷移は，その時点における変数の値，状態，発生したイベントが特定されている．従って，これらのいずれかが異なる場合には不具合とはならない．その為，分岐点に対応する状態遷移において別の状態に遷移したり，変数の値を更新するアクションが異なったりする事で不具合とならず正常な動作を継続できる可能性がある．この事から，不具合の原因は，状態の遷移先が間違っている，アクションによる変数の更新処理が間違っている，或いは状態中のガード条件が間違っている，という 3 つの原因のいずれかであると考えられる．

5. 評価

提案する不具合原因特定手法の有効性を評価する為に，過去にモデル検査を実施した複数の事例を対象に，原因箇所を正しく特定する事ができるかを確認した．評価の為の適用は，事前に不具合の原因が判明している事例を対象に行った．手法により特定された原因箇所が，事前に判明している原因箇所と一致すれば，手法によって正しく原因を特定できたと判断する．適用した事例の規模や反例の数，原因箇所等は，表 1 に示す通り．

表中の探索状態は，検査時に探索した状態遷移の数を表している．また，検査時間は 4.2 節で述べた変更を加えたモデル検査器によって，全ての実行履歴が出力されるまでの時間を表している．特定時間は，原因箇所の特定に要した時間として，4.3 節と 4.4 節で述べた状態空間のグラフ作

表 1 適用実験の結果

評価項目 / 適用対象	プロセス（個）	探索状態（個）	反例（個）	検査時間（秒）	原因箇所（箇所）	特定時間（秒）
事例1	1	2740	110	1.5	1	1.17
事例2	2	2674	108	1.14	30	0.69
事例3	2	40736	515	71.8	110	54.82
事例4	2	50985	153	3.82	2	5.70
事例5	3	41508	156	9.44	24	14.38

成と原因箇所の特定にかかった時間の合計を表している．

提案手法を適用した 5 つの事例では，いずれも事前に把握していた原因箇所を正しく特定する事ができた．また，原因箇所の特定は現実的な時間で完了する事ができ，ソフトウェア開発において提案手法によって原因箇所を特定する事は効果的であると言える．

6. 関連研究

反例の原因を特定する手法としては，並行に動作する複数のプロセスの実行順序によって予想しない処理結果になってしまう race conditions 問題を対象として，原因箇所と修正方法を特定する手法が提案されている[4]. しかし, race conditions 問題以外に用いる事ができない．

また，反例と正常な動作を比較して，反例にのみ存在する状態遷移を原因箇所の候補として挙げる手法[6]や反例と正常な動作の差異にあたる状態遷移を原因箇所の候補とする手法[7]が提案されている．しかし，いずれも原因箇所の候補を示すに留まっている．

SPIN 以外のモデル検査ツールを用いた手法では, モデル検査ツール UPPAAL を用いて原因箇所に相当する条件分岐を特定する手法が提案されている[8]．しかし，UPPAAL には SPIN の様に全ての反例を出力させる機能が無い為，一度のモデル検査で原因箇所を特定する事ができず，原因箇所の特定の為にモデル検査を複数回行う必要がある．

こうした関連研究に対して，本稿の提案手法は扱う問題を限定せず，原因箇所を特定する事が可能である．また，一度のモデル検査によって全ての反例と正常な動作を出力させ，そこから原因箇所を特定できる事から，原因を特定する時間的な効率も優れていると考える．

7. おわりに

7.1 まとめ

モデル検査によって得られる反例の解析と原因の特定を自動化する手法を提案した．これまでは人手によって反例を解析し，正常な動作との差異等から原因箇所を特定していた為，正常な動作を理解している分析者でなければ反例の解析ができない等の問題があった．しかし，提案手法では機械的に原因箇所を特定できる為，誰でも原因の特定が可能になった．

提案手法は複数の事例に適用し，不具合の原因箇所を正しく特定できる事を確認できた．この事から，提案手法が有効である事が分かった．

7.2 今後の課題

提案手法では，不具合の原因箇所を変数の値と状態を表す ID，イベントを表す ID によって識別した．評価の為の試行において，幾つかの事例では複数の原因箇所が特定された．特定された原因箇所に対応する状態遷移では，上記の値が類似していた．この事から，提案手法によって特定された原因箇所の幾つかは，修正すべき誤りとしては同一である可能性がある．そこで，提案手法で特定した原因箇所を相互に比較し, 原因箇所に共通する値を抽出する事で，不具合に直接関係する値を絞り込み，逆に不具合に関係しない値を除外する事ができると考えられる．

今後，特定した原因箇所同士を比較し，不具合に関係する値を更に詳細に特定する事を考えている．

参考文献

1) 高田沙都子, 森奈実子, 村田由香里: モデル検査自動化ツールの開発～検査自動化と反例解析効率化～, 情報処理学会第 74 回全国大会(2012)
2) 森奈実子, 高田沙都子, 長谷川保, 村田由香里, 進博正: モデル検査自動化ツールの開発～入力支援機能と状態遷移表縮約機能～, 情報処理学会第 74 回全国大会(2012)
3) 藤本宏, 森奈実子, 村田由香里: モデル検査における検査対象と外部環境の自動合成手法, 情報処理学会第 76 回全国大会(2014)
4) 陳適, 青木利晃: モデル検査ツールにより出力された反例に基づく誤り特定に関する研究, IPSJ SE-177 No.6(2012)
5) G.J.Holzmann: THE SPIN MODEL CHECKER, Addison-Wesley Professional(2003)
6) Thomas Ball, Mayur Naik, Sriram K. Rajamani: From Symptom to Cause: Localizing Errors in Counterexample Traces, POPL'03(2003)
7) Alex Groce, Willen Visser: What Went Wrong: Explaining Counterexamples, In SPIN Workshop on Model Checking of Software(2003)
8) 青木善貴, 松浦佐江子: モデル検査における反例解析容易化支援, IEICE KBSE2013-79(2014)

ⓒ 2015 Information Processing Society of Japan