制御ルールの並びに着目した反例分析手法の提案

薄井 翔　上田 賀一　小飼 敬　高橋 竜一　堀田 大貴

社会インフラを支えるシステムで障害が起きてしまうと，人命や生活に大きな影響を与えるため，システムの品質保証が必要となっている．近年の大規模・複雑化したシステムの品質を保証するには，従来の方法では対処しきれず，形式手法の一種であるモデル検査が注目されている．モデル検査は，システムの振舞いを網羅的にシミュレートし，期待する性質を満たすか検証する手法である．しかし，モデル検査においては，問題（反例）の発見をすることはできるが，その問題を解決する具体的な修正方法は示されないため，問題箇所の特定が困難である．本研究では，制御ルールに基づき振舞いを起こす情報制御システムを対象とし，反例に到達する制御ルールの並びに着目することで問題箇所を特定する手法を提案した．

Quality assurance of the system is required. Because, when failure occurs in a large-scale control system, it will affect life. It is difficult with the conventional method to guarantee the quality of a large and complicated system in recent years. Therefore, model checking has attracted attention. Model checking is a method of exhaustively simulating the behavior of the system and verifying whether it fulfills the expected properties. However, model checking has problems. That is, don't show concrete solutions when model checking outputs counter example. In this research, we focus on an information control system that behaves based on control rules and focused on the sequence of control rules in order to identify problem areas.

1　はじめに

社会インフラを支える大規模な制御システムで障害が置きてしまうと，人命や生活に大きな影響を与えるため，システムの品質保証が必要となっている．そこで，形式手法の一種であるモデル検査が注目されている．モデル検査とは，検査対象のシステムの振舞い仕様をモデル化し，モデルの取りうる状態を網羅的に探索することで，期待する性質を満たすか検証する手法である．これにより，従来は人手で行っていた工程を機械的かつ網羅的に行うことができるので，大規模システムであっても品質を保証することが可能である．しかし，実務で利用すると検査対象の規模の大きさから，検査が終了しないという状態爆発問題が起きてしまう [1]．さらに，モデル検査による検証結果から実システムの問題箇所を特定することが困難となる問題も存在している．

本研究の先行研究では，1 次処理として状態爆発を防ぐために状態空間を複数の部分に分割し，モデル検査を適用している．2 次処理として空間を状態空間をまたぐような検証を行い，問題があれば反例を出力する．本研究では，検証結果において出力される反例から問題箇所を特定するために，制御ルールの並びに着目した反例分析手法を提案する．先行研究ではデータベースとしてリレーショナルデータベースを利用しているが，本研究ではグラフデータベースの利用についても検討する．

2　先行研究

2.1　情報制御システム

本研究が対象とする情報制御システムは，作業員が手作業で行っていた制御を自動化するために，作業員のノウハウを制御ルールとして体系化し，設備を自

Proposal of counter-example analysis method focusing on the sequence of control rules.

Kakeru Usui, Yoshikazu Ueda, Ryuichi Takahashi, Hiroki Horita , 茨城大学, Ibaraki University.

Kei Kogai, 茨城工業高等専門学校, Ibaraki National College of Technology.

動で制御するシステムである．具体的には，制御プログラムと物理環境から構成され，制御プログラムは制御ルール・仮想設備・仮想センサから構成され，物理環境は設備・センサ・制御対象から構成される．制御用プログラムはセンサから制御対象情報を取得し，取得した情報と制御ルールから設備への操作指示を出力する．情報制御システムは，仮想環境の値を更新し，制御を繰り返すシステムといえる．

2.2 状態爆発問題

モデル検査では，状態を網羅的に探索するため，調べる状態が非常に多くなる可能性がある．状態は属性値の組であるため，属性値の組み合わせ分だけ状態数が存在する．たとえば 2 個の属性がそれぞれ，10 個の属性値を持つ場合，状態は $10^2 = 100$ 状態となるが，属性の数が 2 倍になると，$10^4 = 10000$ 状態となり，状態数が指数関数的に増加する．検査する振舞いによっては，コンピュータのメモリに入りきらないくらいの状態を調べなければならなくなり，モデル検査ツールにより検査できないという問題が生じることがある．この問題を状態爆発問題と呼ぶ．状態爆発問題はモデル検査における解決するべき問題の 1 つとなっている [1].

2.3 段階的検査手法

段階的検査手法とは，状態爆発問題を防ぐために考案されたモデル検査手法である [3]．具体的には，状態空間を分割して小さな領域ごとにモデル検査を適用する．段階的検査手法の処理は 1 次処理と 2 次処理に分かれている．1 次処理では振舞いモデルの分割を行う．分割方法としては，制御ルールによって変化する状態に注目し，制御ルールによる相互関係を少なくするような箇所で分割を行う [2]．2 次処理では，分割された状態空間をまたぐような検証を行う．分割された状態空間を合成して検証を行うのだが，すべての状態に対して合成状態を作成してしまうと状態爆発が起きることが分かっている．そのため，検証内容に応じて必要な部分のみ合成しながら検証を行う．

2.4 グラフデータベース

グラフデータベースとは，データの構造が全てグラフ構造で表されるデータベースである．リレーショナルデータベースとは異なり，NoSQL に分類され，ノード・リレーションシップ・プロパティなどの概念が存在する．実用例としては，SNS における友達の友達を検索したり，ネットショッピングにおける「この商品を買った人はこんな商品も買っている」のようなことを表示するために使われている．このように，2 つのノードの最短距離などの計算が高速に行えることが特徴となっており，もう 1 つの特徴として，クエリ言語の可読性が高いことも挙げられる．代表的なグラフデータベースとして Neo4j があり，Cypher と呼ばれるクエリ言語を使用してグラフを検索する．

3 反例分析手法

3.1 ツールの開発

本研究の先行研究では，反例の原因を単体ルールが原因の場合，ルール不足や実行優先度が原因の場合，ルールの組み合わせが原因の場合の 3 つとしている [4]．これらの原因を調べるために 4 つの検索方法を実装した．

1. **ルール単体検索**
   ルール単体検索では，反例パスと同じルールが正例で適用されているかどうかを検索する．あるルールが反例パス上でのみ適用されている場合は，そのルール単体で反例の原因となっている可能性がある．そのため，本ツールでは反例パス上でのみ適用されているルールが存在するならば，そのルールを出力する．
2. **状態分岐検索**
   状態分岐検索では，反例パス上の状態から正例への状態に遷移させるようなルールが存在するかどうかを検索する．正例への遷移を持たずに反例となる場合はルールが不足している可能性がある．また，反例にならない遷移を持っているにも関わらず反例となってしまう場合は，ルールの実行優先度が原因の可能性がある．そのため，本ツールは反例パスの上の状態からの分岐が存在するならばその状態とルールを出力する．

3. **ルール順序検索**

   ルール順序検索ではパターンマッチングを用いる．まず，反例パスから任意の長さの状態列を抜き出す．その状態列で適用されたルールのリストを R' とする．すべてのパスにおいてパターンマッチングを行い，R' と一致するパターンを持っているパスを検索結果として出力する．反例パスと同じルールが反例パスのみで適用されている場合は，適用ルールの順番に原因がある可能性がある．検索の例として，反例パスから３個取り出した場合 $R' = R_1, R_2, R_3$ となり，検索するルールリストは (R_1, R_2, R_3),(R_1, R_2),(R_2, R_3) となる．

4. **ルール順列検索**

   ルール順列検索では，ルール順序検索でのルールリスト R' 内の要素の組み合わせ R'' をもって，ルール順序検索と同様の検索を行う．つまり，反例パスに適用されているルールを順不同で適用しているパスを検索する．ルール順序検索と同様に，適用ルールの組み合わせに原因がある可能性がある．検索の例として，$R' = R_1, R_2, R_3$ の場合の検索するルールリストは R'' =(R_1, R_2, R_3),(R_1, R_3, R_2),(R_2, R_1, R_3), (R_2, R_3, R_1),(R_3, R_1, R_2),(R_3, R_2, R_1) となる

次に使い方を説明する．まず，ルール単体検索を行い，ルール単体で反例の原因になっているかを調べる．次に状態分岐検索を行い，反例パスの構造を把握する．最後にルール順序検索・ルール順列検索を行い，反例パスでのみ適用されているルールの並びを調べる．反例パスでのみ適用されているルールの並びが存在すれば，それらを確かめることで反例の原因を特定する手がかりになると考える．

また，実行時間の比較を行うために，利用するデータベースが異なり同じ機能を持つツールを２通り実装した．ただし，リレーショナルデータベースである MySQL を使用する場合，ルール順序検索・ルール順列検索のようなグラフを走査するような探索の場合は遅くなることが既に分かっている．そのため，文字列化処理を行う．これは，データベースに保存されているすべてのパスを文字列としてメモリに確保し，その文字列を検索することで実行時間を早めるものである．

表 1　実験環境

OS	Yosemite
メモリ	16GB
CPU	2.5 GHz intel Core i5
Java	1.8.0_40
Python	2.7.11
モデル検査ツール	SPIN Version 6.4.5

4　適用実験

本研究の対象である制御システムにおいて本ツールを使用し実験を行い，反例の原因を特定できるかどうか確かめる．また，２つのツールの実行時間の比較を行う．今回の実験で使用する反例パスについては，検査項目に違反している遷移にマーキングを行い，初期状態からその遷移をしたパスを別ファイルとして保存し，到達可能性について検査した結果として扱う．実験環境について表 1 に示す．

4.1　適用事例

情報制御システムの一種であるエレベータの制御システムを適用事例とする．このシステムは，呼び出しボタンや階指定ボタンの情報から建物内のエレベータの動きを制御し，円滑にエレベータを移動させるシステムである．今回実験に適用したエレベータ制御システムは，エレベータと各フロアに取り付けられた呼び出しボタンから構成されており，建物の階数が３階，エレベータが１つ，定員は１人となっている．エレベータ情報を表 2，フロア情報を表 3，制御ルール一覧を表 4 に示す．また，エレベータのステートマシン図を図 1 に示す．

違反する検査項目については，エレベータが定員オーバーのまま移動してしまうようなモデルを作成した．全体として，状態数 247,040 個，遷移数 232,000 個となった．

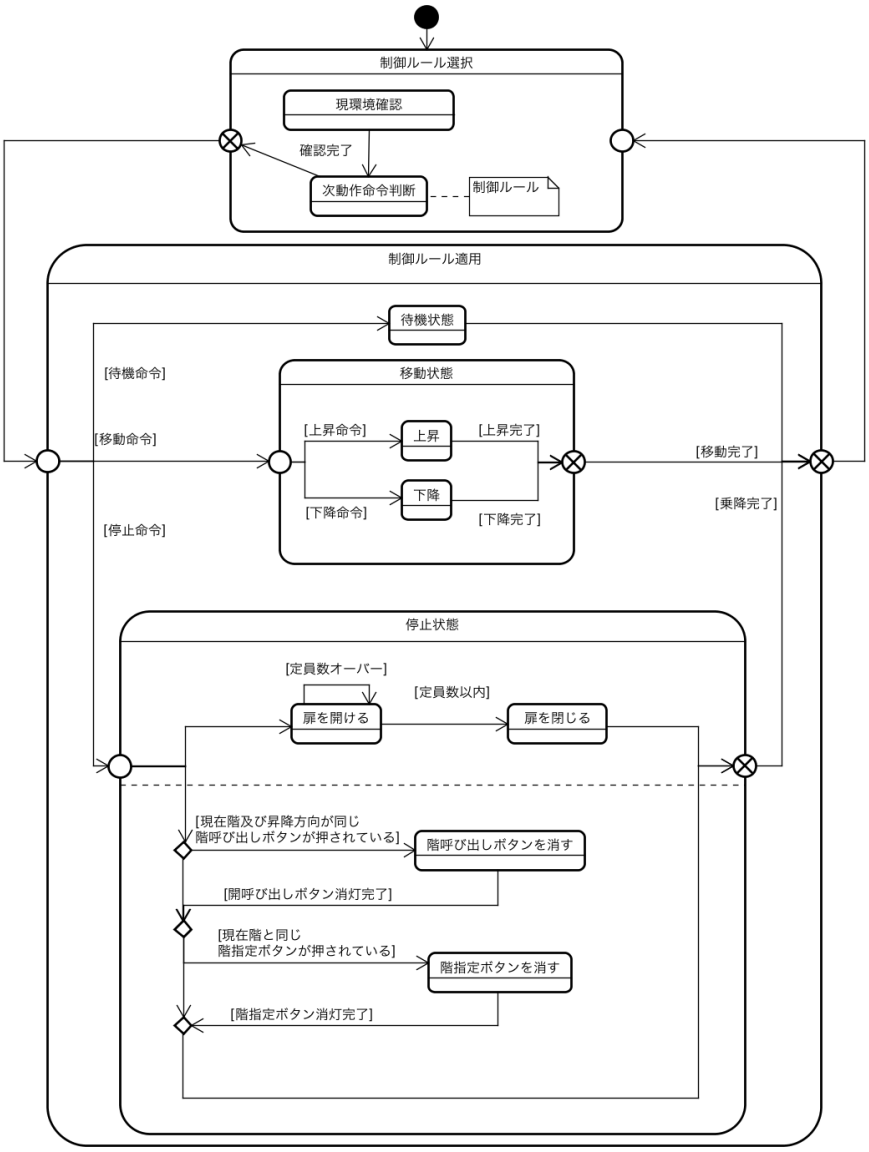

図 1　エレベータのステートマシン図

4.2　実験結果

４つの検索結果について以下に示す．また，２つのツールはどちらも同じ結果となる．

- ルール単体検索

反例パス上のみに適用されているルールは存在しなかった．そのため，ルール単体が反例に影響していないことが考えられる．

- 状態分岐検索

表 2　エレベータ情報

属性	説明
階層	現在の階層
乗員	定員オーバーかどうか
状態	移動中 or 停止中
目的階	目的の階層
階指定ボタン	押されているかどうか
昇降方向	上昇 or 下降 or なし
扉	開いている or 閉じている

表 3　フロア情報

属性	説明
呼び出し上ボタン	押されているかどうか
呼び出し下ボタン	押されているかどうか

反例パス上から正例へのパスは存在しなかった．すべての反例パスが分岐せずに初期状態から反例へ遷移していた．そのため，反例から正例に遷移するようなルールが不足している可能性が考えられる．

- ルール順序検索

  反例パス上にしか存在しないルールの適用順序を発見した．

  ルールが (1,6,9,5,8,19,19,17,2,5,8) の順番で適用されているパスは反例パスのみとなっていた．表 3 より，具体的な動作を確認する．エレベータ内の乗員が現在より下の階指定ボタンを押し，下降を開始している．しかし，ルール 17 によって既に乗っていた乗員の目的階に到達する前に，待機している人がいる階で停止し，扉を開けている．1 人目が降りていない状態で 2 人目が乗ったまま扉が閉まり，エレベータが下降しようとしていることが分かる．また，上昇中においても同じ現象が反例上で起きていた．反例パスのみに存在する最短のルールリストは (2,5,8),(2,4,7) となった．

- ルール順列検索

  ルール順序検索の結果である (2,5,8),(2,4,7) について検索を行った結果，組み合わせを変えた場合については正例と反例のどちらにも存在しなかった．

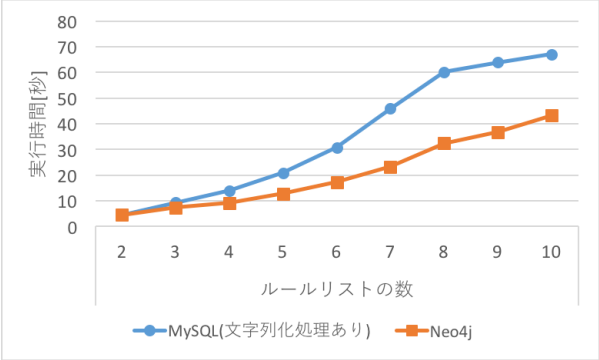

図 2　ルール順序検索の実行時間 [秒]

以上から，ルールを (2,5,8),(2,4,7) の順番で適用した場合に反例となる可能性が高いことが考えれれる．また反例になる前の遷移にルール 7 やルール 8 が適用されている．これらのルールはドアを閉めるルールであるため，ドアを閉める処理をする前に，定員オーバーになっているかを判定する処理を追加することで，反例になることを防ぐことができると考えられる．

4.3　実行時間の比較

4.2 節の実験をするにあたり，実装した 2 つのツールの実行時間の比較を行った．また，MySQL を利用する場合においては，3.1 節で記述した通り，文字列化処理を行った場合についても比較を行った．今回のモデルについては文字列化処理に約 37 分かかった．実行時間が極端に長くなるものについては実験を中断した．

ルール単体検索・状態分岐検索の実行時間を表 5，ルール順序検索の実行時間を表 6，ルール順列検索の実行時間を表 7，それぞれのグラフを図 2，図 3 に示す．

表 4　エレベータの制御ルール一覧

ルール番号	適用条件	適用ルール
1	階指定ボタンが押された階に到着	ドアを開ける
2	呼び出し上ボタンが押されている階に到着	ドアを開ける
3	呼び出し下ボタンが押されている階に到着	ドアを開ける
4	エレベータが上昇開始	上昇開始した階の呼び出し上ボタンを消す
5	エレベータが下降開始	下降開始した階の呼び出し下ボタンを消す
6	目的階に到着	目的階の階指定ボタンを消す
7	昇降方向が上昇になる	ドアを閉める
8	昇降方向が下降になる	ドアを閉める
9	目的階が現在階より下	エレベータの昇降方向を下降にする
10	目的階が現在階より上	エレベータの昇降方向を上昇にする
11	エレベータが停止中に呼び出し上ボタンが押された	呼び出し上ボタンが押された階をエレベータの目的階にする
12	エレベータが停止中に呼び出し下ボタンが押された	呼び出し下ボタンが押された階をエレベータの目的階にする
13	目的階より上の階で呼び出し上ボタンが押された	エレベータの目的地を呼び出し上ボタンが押された階に変更する
14	目的階より下の階で呼び出し下ボタンが押された	エレベータの目的地を呼び出し下ボタンが押された階に変更する
15	目的階に到着	エレベータの情報をリセットする
16	上昇中に階指定ボタンが押された階，または呼び出し上ボタンが押された階に到着	エレベータを停止する
17	下降中に階指定ボタンが押された階，または呼び出し下ボタンが押された階に到着	エレベータを停止する
18	昇降方向が上昇になる	エレベータが上昇する
19	昇降方向が下降になる	エレベータが下降する
255	適用するルールが存在しない	変化なし

表 5　ルール単体検索・状態分岐検索の実行時間 [秒]

	Java+MySQL	Python+Neo4j
ルール単体検索	45.3	12.0
状態分岐検索	43.6	1315.0

4.4　考察

反例が存在するモデルに対して実験を行った結果，反例の原因を見つける手助けを行うことができた．反例パスのみに適用されているルールリストを見つけることができれば，反例の原因を特定できると考えられる．実行時間については，文字列化処理を考慮した場合，状態分岐検索以外においてグラフデータベースである Neo4j を利用した方が速いことが分かった．

表 6　ルール順序検索の実行時間 [秒]

ルールリストの数	Java+MySQL		Python+Neo4j
	文字列化処理なし	文字列化処理あり	
2 個	3193.9	4.5	4.5
3 個	4878.8	9.3	7.4
4 個	20413.7	14.0	9.2
5 個	実行中断	20.9	12.8
6 個	実行中断	30.7	17.3
7 個	実行中断	45.9	23.3
8 個	実行中断	60.2	32.4
9 個	実行中断	63.9	36.8
10 個	実行中断	67.0	43.2

表 7　ルール順列検索の実行時間 [秒]

ルールリストの数	Java+MySQL		Python+Neo4j
	文字列化処理なし	文字列化処理あり	
2 個	10933.2	1.5	4.3
3 個	実行中断	2.3	3.9
4 個	実行中断	1.1	12.8
5 個	実行中断	2.5	69.4
6 個	実行中断	2.0	409.4
7 個	実行中断	9.9	2844.0
8 個	実行中断	354.0	実行中断

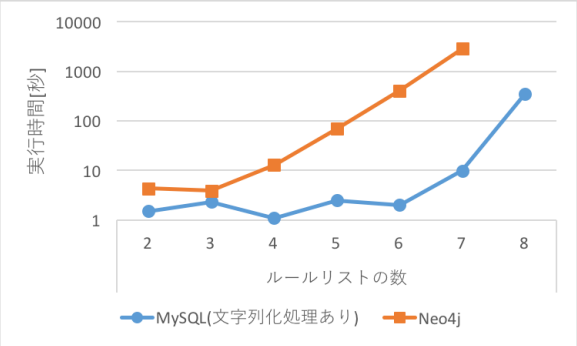

図 3　ルール順列検索の実行時間 [秒]

リレーショナルデータベースにおいてグラフ構造をたどるには，1 つの遷移を調べるごとに遷移後状態を遷移前状態として検索することを繰り返すためクエリが複雑になり，さらにループ判定を行う必要があるため処理が遅くなってしまうことが分かる．一方，グラフデータベースでは 1 つのクエリで検索を行うことができるため，処理が高速になっていることが分かる．今回実装した機能はグラフ構造を把握する必要があるため，グラフデータベースである Neo4j が速い結果となった．しかし，状態分岐検索では 1 つの遷移ごとに検索を行うため，1 つの遷移ごとに格納されているリレーショナルデータベースである MySQL の方が速い結果となった．データベースによる差が大きかったため，実装言語の差については比較を行っていない．

5　まとめ

本研究では，制御ルールの適用順序に着目した反例分析ツールを開発し，エレベータ制御システムの

モデルに対して実験を行った．また，利用するデータベースの違いについて比較を行った．反例パスにおけるルールの順序に着目することで，反例の原因特定の手助けを行えることを示した．また，グラフ構造のデータを解析する場合はリレーショナルデータベースよりもグラフデータベースの方が優秀であることを示した．今後は，他のモデルに対して実験を行い，検査者に対してさらにわかりやすく結果を出力できるように実装を行う．

謝辞

本研究を進めるにあたり，ご協力くださった (株) 日立製作所インフラシステム社の武澤隆之様，山形知行様，テクノロジーイノベーションセンタの奏野康生様に深く感謝いたします

参 考 文 献

[1] 吉岡信和, 青木利晃, 田原康之: SPIN による設計モデル検証, 2008.

[2] 古木隆裕, 小飼敬, 上田賀一: 情報制御システムのモデル検査におけるモデル分割支援法の検討, 日本ソフトウェア科学会大会論文集, Vol. 33(2016), pp. 453–460.

[3] 小山恭平, 小飼敬, 上田賀一: 情報制御システムに対する SPIN を用いた段階的モデル検査手法, 日本ソフトウェア科学会大会論文集, Vol. 30(2013), pp. 274–281.

[4] 大森祐貴, 小飼敬, 上田賀一: 段階的検査法を用いたモデル検査の反例分析手法, 日本ソフトウェア科学会大会論文集, Vol. 32(2015), pp. 6p.