Vol.2018-SE-198 No.9
2018/3/9

グラフデータベースを用いたモデル検査手法の提案

久野 和敏[1]　上田 賀一[1]　小飼 敬[2]

概要：モデル検査において，検査対象の状態数が膨大である場合，現実的な時間で検証できないという問題があり，実用化の課題となっている．先行研究ではこの問題を軽減するため，結合度による振舞いモデルの自動分割と部分状態遷移の生成によって，検査対象の状態数の削減に取り組んだ．しかし，この手法で生成された状態遷移は情報が部分集合化されるため，既存のモデル検査器では検証できない問題が生じた．本研究では，先行研究を用いたモデル検査において，グラフデータベース Neo4j の問い合わせと最小限の状態合成によって，既存のモデル検査器が行う検証と同等な検証を行うための手順を提案する．シミュレーション実験の結果，提案手法による検証が行えることを確認した．

キーワード：モデル検査, グラフデータベース,Neo4j

A Model Checking Approach Using Graph Database

Kazutoshi Kuno[1]　Yoshikazu Ueda[1]　Kei Kogai[2]

Abstract: In model checking, when the number of states of an object to be inspected is enormous, there is a problem that it can not be verified with realistic time, which is a problem of practical application In previous research, in order to alleviate this problem. we tried to reduce the number of states to be inspected by automatic segmentation of behavior model by degree of coupling and generation of partial state transition. However, since the information is subsetted in the state transition generated by this method, there is a problem that can not be verified by the existing model checker. In this study, we propose a procedure for verification equivalent to verification performed by an existing model checker with by inquiries of the graph database Neo4j and minimal state synthesis of model checking using previous research. As a result of simulation experiments, we confirmed that we can verify by the proposed method.

Keywords: model checking,graph database,Neo4j

1. はじめに

情報システム，とりわけ社会インフラを支えるようなシステムにおいて，1 つの誤りが引き起こす損失や影響力は非常に大きい．そのため，これらのシステムには高い安全性や信頼性を持つことが要求される．このような課題に対し，アルゴリズム的に検証を行う方法として形式手法の 1 つであるモデル検査がある．モデル検査とは，システムの振舞いを抽象化した仕様記述 (モデル) を元にシステムの取り得るすべての状態を網羅的に探索することでシステムに求める特性が満たされているか検証する手法である．モデル検査を適用するにあたり，考慮しなければならないのが状態爆発問題である．状態爆発問題とは，生成された状態数が膨大な数である場合，検証処理が現実的な時間内に終わらず，検査不能となる問題である．そのため，状態爆発問題はモデル検査の実用化における課題となっている．

状態爆発への対策として，小山ら [1]，宮島ら [2] による

[1] 茨城大学
Ibaraki University
[2] 茨城工業高等専門学校
Ibaraki National College of Technology

研究として段階的モデル検査手法が提案された．段階的モデル検査とは，モデル検査を 2 つの段階に分けたものであり，1 次処理でモデルを分割し，2 次処理で分割されたモデル間をまたぐ全ての状態の生成を行い状態探索を行うことで，1 度の検証で探索する状態変数の数を抑える手法である．古木ら [3] による先行研究では，モデルに記述された振舞いどうしの結合度を元に，モデルを手順的に分割することで適用者の知見や経験に依らない状態空間の分割手法を提案した．既存のモデル検査や前述の段階的モデル検査との違いとして，1 次処理で分割した状態空間を合成することなくデータベースに格納する点が挙げられる．また，分割されたモデルを元に生成された各部分状態は情報が符号化されている．この 2 点によって，既存のモデル検査器では検証が行えない．そこで本研究では，既存のモデル検査器が行っている検証として，SPIN モデル検査器に用いられる線形時相論理式を用いた検証と同等な検証を先行研究を用いたモデル検査で行うための手順を提案する．グラフデータベース Neo4j に格納された部分状態への問い合わせによって状態空間の探索を行う．各部分状態集合への問い合わせによって得られた部分遷移に対して，先行研究で提案された分割された状態を元のモデルでの状態に複合する手法を適用し，複合した状態に対して検証条件を満たすかどうかを確認することで検証を行う．この手法を各時相論理に適用し，正例，または反例となる状態までの遷移を示すことで，DB への問い合わせを用いて検証を行う．

2. 段階的モデル検査

ここでは先行研究として提案された段階的モデル検査について述べる．

2.1 概要

小山ら [1]，宮島ら [2] による先行研究によって，段階的モデル検査が提案された．段階的モデル検査とは，2 つのアプローチを元に状態爆発を低減する手法である．1 つは，検査対象のモデルを分割し 1 度に生成する状態数を狭めることによる状態作成の処理時間を削減する手法である．もう 1 つは 振舞いモデルに記述される属性に着目し，検査項目に関係する属性のみを用いて状態空間を生成することにより，検査対象となる状態数を削減する手法である．これら 2 つのアプローチの実現のためにモデル検査の処理を 2 段階に分割したことから，段階的検査法と呼ばれる．検査対象のモデルを分割するアプローチにおいて，幾つかの基準が提案されていたがいずれも検証者自らによる分割が必要であった．古木らの先行研究 [3] ではこの問題に対し，人手での導出が特に困難である振舞い結合度によるモデル分割の自動化が行われた．振舞い結合度での分割とは，システムの振舞いどうしの影響度合いを元にモデルを分割する手法であり，分割の方針として元のモデルから 2 つの部分モデルを作成すること，部分モデル間の依存関係の増加を抑えつつ，部分モデルの大きさの偏りを減らすことが挙げられた．この手法による段階的モデル検査の概要図を**図 1** に示す．

2.2 手順

検証を行う手順として，まず検査対象の情報制御システムを元に仮想的に表現した振舞いモデルを作成する．先行研究の手法により，振舞いモデルに記述された属性定義と制御ルールの情報を元にモデルが分割され，システムの振舞いが部分的に表現された部分モデルが 2 つ作成される (部分モデル A，部分モデル B)．

次に 2 つの部分モデルの情報を元に，モデル検査器 SPIN を用いてそれぞれの状態空間の生成が行われる．生成された状態集合は部分モデルとの対応をとり，部分遷移集合 A，部分遷移集合 B と呼ばれ，1 つの部分遷移は前状態，後状態，適用された制御ルールの 3 つの情報で構成される．最後に，生成した部分状態はグラフデータベース Neo4j に格納され，検証は Neo4j への問い合わせを用いて行われる．

2.3 全体状態と部分状態

モデルの分割によって，生成される状態遷移は**図 2** のように変わる．モデルを分割せずに作成した時の状態を全体状態と呼び，それらの遷移関係をまとめたものが全体遷移集合である．また，部分モデルを元に作成された状態を部分状態と呼び，それらの遷移関係をまとめたものが部分遷移集合である．図 2 上部では全体状態 S_0 が制御ルール R_0 によって全体状態 S_1 に状態遷移しているが，モデル分割により部分状態 A_0 が制御ルール R_0 によって部分状態 A_1 に状態遷移するようになる．このように全体状態では全ての属性についての情報を持つが，モデルの分割により属性定義と制御ルール定義の記述が分割されているため，各部分状態では断片的な情報しか持たない．どの部分状態がどの属性の情報を持つかは符号化テーブルに記述され，状態生成時や検証時に使用される．“*”は符号化された属性を示し，その部分モデル上ではその属性については関与しない．そのため，符号化された属性に対する制御ルールはその部分モデル上には存在しない．このように各部分状態モデルが持つ属性の数が減少するため，モデル検査器で生成する状態数の総数を減らすことができる．

2.4 状態の復号

分割されたモデル上ではそれぞれに対応した属性の変化にしか関与しないため，検証条件として指定された属性がそれぞれの部分モデルの属性を参照しなければならない場合，片方の部分状態集合への状態探索だけでは検証できず，両方の部分モデルを横断するような状態探索を行う必要がある．全ての全体状態とその遷移を作成することは状

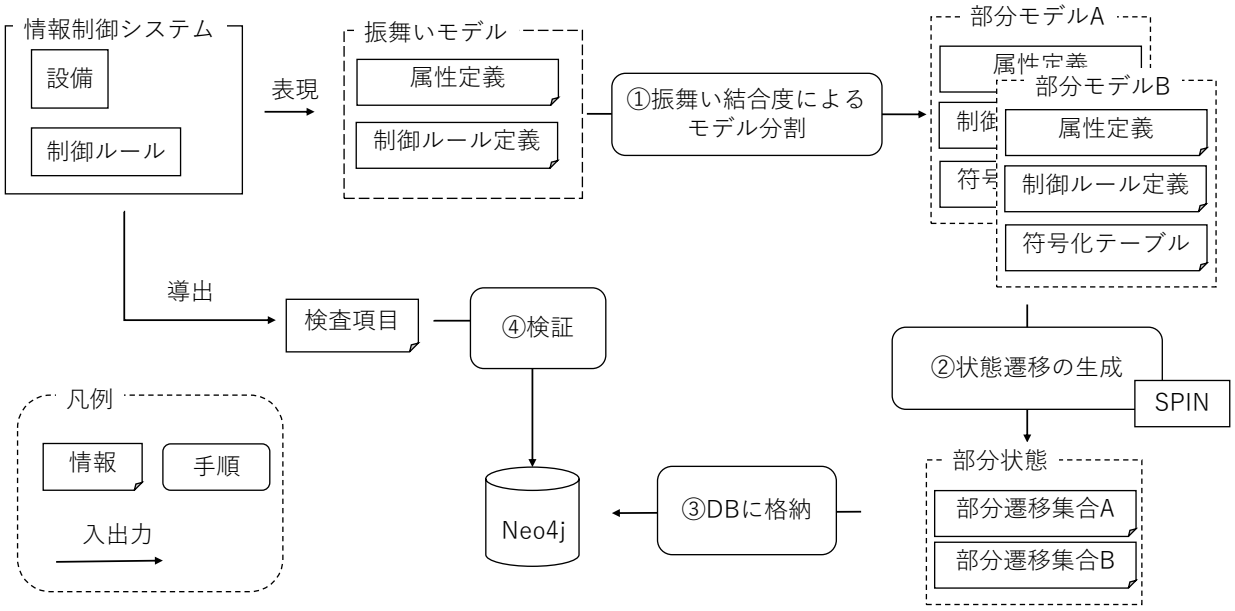

図 1　段階的モデル検査の概要図
Fig. 1　Outline of stepwise model checking

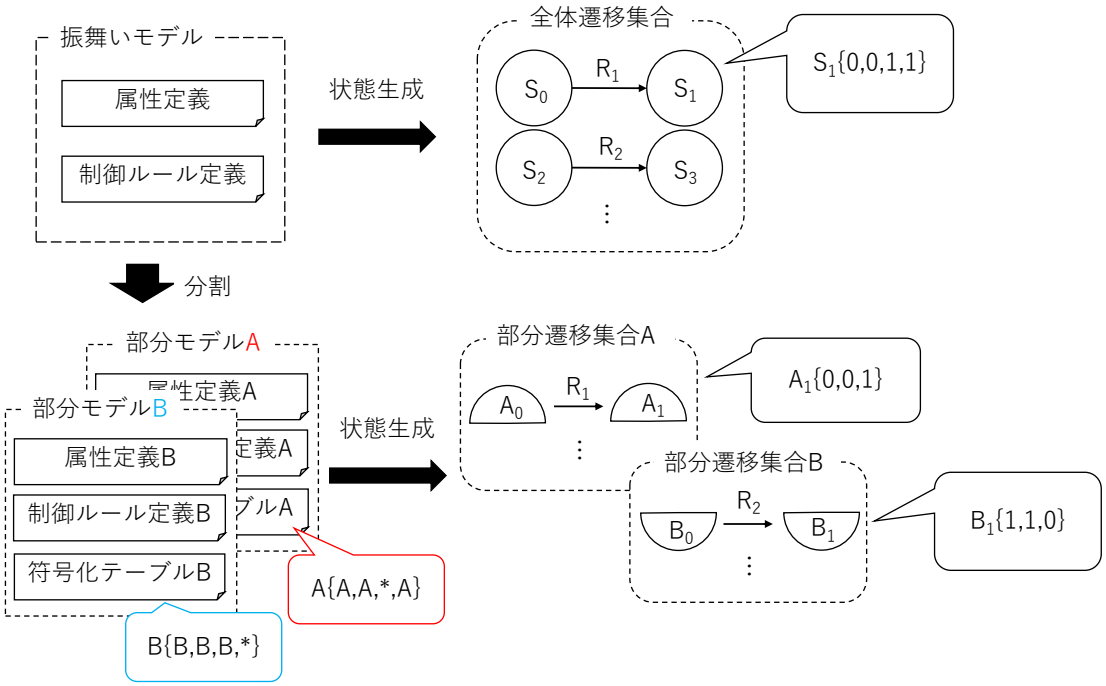

図 2　モデル分割による状態の符号化
Fig. 2　Encoding states by partitioning the models

態爆発を招くため，部分状態集合への状態探索中に状態の符号化と復号を行うことで最低限の全体状態を作成しながら検証を行う．探索条件としての初期状態を全体状態で指定し，この情報を元に復号を行うことで全体状態での検証を行い，結果として全体状態での状態遷移パスを出力できるようにする．この手法の概要を図 2 に示す．

手順として，まず現在の状態である S_0 の持つ属性の情報を符号化テーブルに従い符号化する．2 番の処理では，1 番の処理によって得られた部分状態の次の部分状態をデータベースに問い合わせる．問い合わせ結果として部分状態と制御ルールを 1 対 1 の組み合わせで取得する．取得した探索結果のうち 1 つの結果を選択し (今回は A_1，R_1)，1 番で使用した符号化テーブルと同じものを用いて復号する．復号した全体状態は “*” が残る不完全な全体状態のため，4 番の処理で復号時に残る “*” の箇所の属性の情報を補完し，全体状態 S_1 を得る．情報の補完は前状態である S_0 の

“*”の箇所の属性を対応させることで行う．これは，符号化テーブルに記述された “*”の属性はその部分状態では値は更新されないため行える．S_0 とこれらの処理で得られた S_1 と R_1 の組み合わせが全体状態の 1 遷移に相当する．

図 2 では部分状態 A に対しての問い合わせのみを行っているが，実際は部分状態 B に対しての問い合わせも行い，その際には符号化テーブル B が使用される．部分状態 A，部分状態 B という手順で問い合わせを行い，探索結果に複数の候補が出る場合の選択方法として先に得られた部分状態から選択する深さ優先探索で行っている．そのため，どの部分状態が先に選択されるかはデータベースの処理手順に依存する．

2.5 Neo4j

Neo4j[6] は java で実装されたオープンソースのグラフデータベース管理システムである．リレーショナルデータベースとは違い，各目的に応じて最適化されたデータベースである NoSQL に分類され，Neo4j はグラフ論理に特化したデータベース管理システムである．広く知られている MySQL[5] 等の関係データベース管理システムとの違いとして，データの構造がネットワーク上になっている場合でのデータ管理，検索に優れている．モデル検査で使用される状態遷移モデルは，状態から状態への遷移をネットワーク構造として表せるため，状態遷移の探索効率のために Neo4j のグラフ探索の機能を用いる．

3. 提案手法

先行研究では全体状態での属性を全て指定することによって，初期状態と終了状態となる状態を 1 つずつ指定し，デッドロックに到達するかの検証方法が提案された．

本研究では，先行研究でも用いられたモデル検査器 SPIN が使用する LTL 式を用いた検証と同等な検証を DB 検索によって行えるようにすることを方針とし，以下の 2 点において拡張した．

(1) 論理式による探索条件の指定
(2) LTL 式の単項式 1 つを用いた場合の検証

探索対象となる状態を論理式で指定できるように拡張した上で, 探索アルゴリズムとして時相論理 Next,Finally,Globally のいずれかの検証をグラフデータベース Neo4j への問い合わせを用いて行う．先行研究同様，状態の探索においては深さ優先探索で行い，状態探索において現在の状態を評価するときにのみ状態を複合することで，最小限の復号を行う．

3.1 条件式の評価

また，検証式の評価には Neo4j の機能を用いて以下の手順で行う．

(1) 復号した状態を検査対象とする
(2) 検査対象を Neo4j に登録する
(3) 検査対象が条件式を満たすか問い合わせを行う

そのため，探索条件として記述される論理式は Neo4j の構文に従い，属性等の記述内容はモデルの記述に依存する．**図 4**，**図 5**，**図 6** にそれぞれの探索アルゴリズムの疑似コードを示す．

4. 適用実験

本章では，提案手法による DB 検索を用いた検証によって，検証条件を検査できるかを確認する．先行研究の手法を用いて状態生成とデータベース Neo4j への状態格納を行う．

4.1 実験環境

実験を行う環境を**表 1** に示す．

表 1 実験環境
Table 1 Experimental environment

実験環境	バージョン
OS	OS X El Capitan 10.11
CPU	Intel Core i5(1.4GHz)
メモリ	16GB
Neo4j	3.2.2

4.2 適用事例

適用事例として立体倉庫制御システムのモデルを使用した．立体倉庫制御システムとは，倉庫への荷物の搬入と搬出の作業を自動で行うシステムである．本研究で使用するモデルでは，2 つのクレーンが立体的に並ぶ棚に対して荷物の運搬を行う．このモデルに対して幾つかの大きさのモデルを用意した．例として**図 7** に 2 階 3 列のシステムの概要図を示す．各棚には 1 つの荷物を配置でき，搬入口では荷物が配置されていない場合に任意のタイミングで新しく荷物を配置することで，搬入が行われる．搬出口も同様に荷物が配置されている場合に任意のタイミングで荷物を搬出する．クレーン 1 つが同時に運搬できる荷物は 1 つであり，棚と同じ位置に存在する場合の荷物の配置，回収が行える．クレーンは一度の移動で上下左右いずれかの棚に 1 つ移動することができるが，すでにその列に別のクレーンがある場合はその列への移動ができない．つまり，図 7 では青のクレーンが B 列に存在するため，赤のクレーンは B 列への移動ができない．

本研究では，実験のために搬出に関する制御ルールを取り除いた振舞いモデルを使用する．搬出に関する制御がなくなることで，最終的に全ての棚とクレーンに荷物が配置され，クレーンの移動のみが可能となる状態に到達する．モデルに記述される振舞いは以下のようになり，立体倉庫

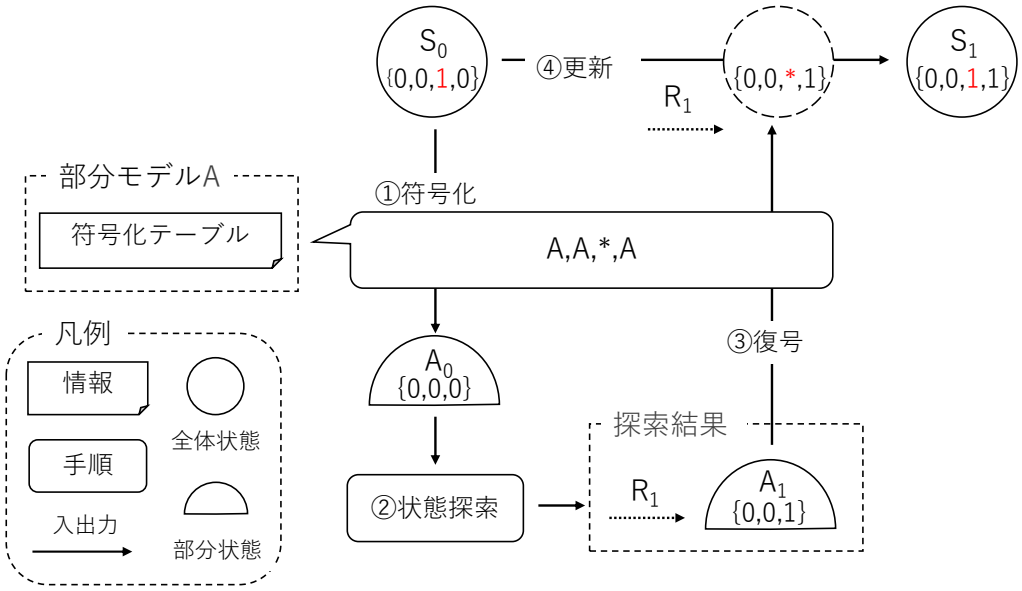

図 3 部分状態の反映
Fig. 3 Decoding partial state

```
初期状態を部分化し，部分状態 A と部分状態 B を取得
部分状態 A の次状態を問い合わせ
部分状態 B の次状態を問い合わせ
問い合わせで得た部分状態を全て全体状態に合成
合成した全体状態を検証対象の状態として DB に登録
検証対象の状態集合に条件φを満たさない状態を問い合わせ
if(存在する)
    発見した状態を判例として出力し終了
else
    検証対象の状態集合に条件φを満たす状態を問い合わせ
    if(存在する)
        発見した状態を正例として出力し終了
    else
        Next(φ) を満たす状態は存在しないので終了
```

図 4 時相論理 Next の問い合わせ手順
Fig. 4 Procedure of LTL Next

```
状態遷移を格納するスタックを定義
スタックに初期状態を格納
While(スタックが空でない){
 スタックの先頭要素を現在状態に設定
 現在状態を符号化し，部分状態 A と部分状態 B を取得
 部分状態 A の次状態を問い合わせ
 部分状態 B の次状態を問い合わせ
 2 つの問い合わせ結果を探索候補に設定
 for(探索候補が空でない){
   探索候補を 1 つ取得
   取得した探索候補を全体状態に合成し，次状態に設定
   if(次状態が探索済みでない){
     次状態をスタックに登録
     For ループを抜ける
   }
 }
 if(次状態が条件を満たすか)
   正例となる遷移パスを見つけたので探索終了
 if(選択可能な探索候補が存在しない)
   スタックの先頭要素の取り出し
}
```

図 5 時相論理 Finally の問い合わせ手順
Fig. 5 Procedure of LTL Finally

制御システムの各要素の取り得る状態としての属性記述をソースコード**図 8** に，上記の振舞いを網羅した制御ルールをソースコード**図 9** に示す．

(1) 搬入口への荷物の搬入する
(2) 赤クレーンの A 列，B 列への移動する
(3) 青クレーンの B 列，C 列への移動する
(4) クレーンが持つ荷物を各棚に配置する
(5) 各棚にある荷物をクレーンが持ち出す

4.3 有効性の評価

提案した各アルゴリズムの妥当性を確認する．

4.3.1 時相論理 Next

時相論理 Next の探索を行うアルゴリズムを確認する．ここではクレーンの動作について検証する．図 7 のように赤のクレーンと青のクレーンが隣り合っている場合，次の状態でクレーンが同じ列に存在するような状態が存在しないのか検証を行う．実験に使用した検証条件とその結果を**図 10** に示す．

not(val7=“0”) によって「赤のクレーンが B 列に来ることはない」という意味になる．つまり Next，not(val7=“0”) は「1 つ後の状態は全て赤のクレーンが B 列に存在しない状態である」という意味になる．また，反例が 1 つでも存在すれば，赤のクレーンが青のクレーンと同じ列にあることになる．実験結果として，反例は出力されず正例が出力された．

```
初期状態が条件を満たしているか確認
初期状態 S の部分状態 Sa と部分状態 Sb を取得
部分初期状態 Sa から部分条件 not(φ a) までの遷移を問い合わせ
部分初期状態 Sb から部分条件 not(φ b) までの遷移を問い合わせ
if(問い合わせ結果が存在する)
 $G(φ)$は成り立たないので探索終了
状態遷移を格納するスタックを定義
スタックに初期状態を格納
While(スタックが空でない){
 スタックの先頭要素を現在状態に設定
 現在状態を符号化し，部分状態 A と部分状態 B を取得
 部分状態 A の次状態を問い合わせ
 部分状態 B の次状態を問い合わせ
 2 つの問い合わせ結果を探索候補
 for(探索候補が空でない){
   探索候補を 1 つ取得する
   取得した探索候補を全体状態に合成し，次状態に設定
   if(次状態が探索済みでない){
     次状態をスタックに登録
     For ループを抜ける
   }else{
     $ G(φ)$が成り立つ遷移パスを見つけたので終了
   }
 }
}
```

図 6　時相論理 Globally の問い合わせ手順
Fig. 6　Procedure of LTL Globally

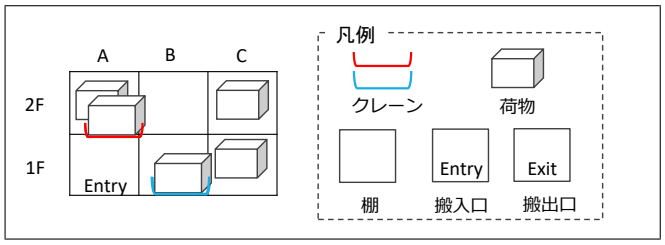

図 7　立体倉庫システムの概要図
Fig. 7　Outline of warehouse system

```
A1, Loading, None
A2, Loading, None
B1, Loading, None
B2, Loading, None
C1, Loading, None
C2, Loading, None
CR1_L, False, True
CR1_X, A, B
CR1_Y, _1, _2
CR2_L, False, True
CR2_X, B, C
CR2_Y, _1, _2
```

図 8　属性記述
Fig. 8　Description of attributes

4.3.2　時相論理 Finally

時相論理 Finally の探索を行うアルゴリズムを確認する．“or”を用いた条件式を使用し，どちらかを満たす状態を発見した時点で探索を正しく終了しているかを確認した．実験に使用した検証条件とその結果を**図 11** に示す．条件式「(val6=“1” and val7=“0” and val8=“1”) or (val9=“1” and val10=“0” and val11=“1”)」は「赤のクレーンが荷物を持った状態で 2F_A に存在する」あるいは「青のクレーンが荷物を持った状態で 2F_B に存在する」ことを意味し，どちらかの条件を満たす状態が発見されれば，その状態までの遷移パスが出力される．実行結果より，赤のクレーンに関する条件は満たすが，青のクレーンに関する条件は満たしていない．検証条件の通りにどちらかの検証条件を満たした時に探索を終了できることが確認できた．

表 2　状態空間の異なるモデルへの適用結果
Table 2　Application to different models of state space

階層数と列数	状態数 (部分 A)	状態数 (部分 B)	処理時間
2 階 3 列	256	256	3s
3 階 4 列	27,648	27,648	22m 18s
3 階 5 列	393,216	393,216	72h 49m 28s

4.3.3　時相論理 Globally

時相論理 Globally の探索を行うアルゴリズムを確認する．「棚とクレーンに全て荷物が設置されている状態の後は，全ての遷移において常に荷物が置かれているか」を確認することで，排出口に関する制御ルールを正しく取り除けているかを検証する．実験に使用した検証条件とその結果を**図 12** に示す．判例が出力されず，正例となる遷移パスが 1 つ出力されたため，排出口に関する制御ルールを正しく取り除けていることが確認できた．この正例では，赤のクレーンが移動する制御ルールによってループが起きる状態遷移を示しているため，その後の状態は全て検査条件を満たしている．

4.3.4　性能評価

さまざまなサイズの立体倉庫制御システムに対して探索を行い，実行時間の差を確認した．図 7 は 2 階 3 列の立体倉庫制御システムであり，この階層や列数を変化させることでモデルに記述されるシステムの属性や振舞いの数を増減させた．探索条件として倉庫内に荷物がない状態から，全ての棚とクレーンに荷物が設置されている状態までを FInally のアルゴリズムによって探索する．各モデルを元に先行研究の手法を用いてモデル分割と部分状態の作成を行った．4 階 5 列のモデルは部分モデルの作成に 10 日程必要な計算量になるため，検証での状態探索が現実的な時間までに終了しないと判断し，3 階 5 列のモデルを作成した．**表 2** に各モデルの状態空間の大きさと実行時間の結果を示す．それぞれモデルとなる倉庫のサイズと生成された部分状態数，条件を満たす状態を発見するまでの処理時間の合計を示している．

4.3.5　考察

各アルゴリズムの実験により，提案手法による DB への問い合わせ手順で LTL 式の時相論理を用いた条件式の検査が行えることが示せた．表 2 による状態空間の異なる状

```
ルール 0:A1=None ->  A1=Loading
ルール 1:CR1_L=True && A1=None && CR1_X=A && CR1_Y=_1 -> CR1_L=False && A1=Loading
ルール 2:CR1_L=False && A1=Loading && CR1_X=A && CR1_Y=_1 -> CR1_L=True && A1=None
ルール 3:CR1_L=True && B1=None && CR1_X=B && CR1_Y=_1 -> CR1_L=False && B1=Loading
ルール 4:CR1_L=False && B1=Loading && CR1_X=B && CR1_Y=_1 -> CR1_L=True && B1=None
ルール 5:CR1_Y=_1 -> CR1_Y=_2
ルール 6:CR1_L=True && A2=None && CR1_X=A && CR1_Y=_2 -> CR1_L=False && A2=Loading
ルール 7:CR1_L=False && A2=Loading && CR1_X=A && CR1_Y=_2 -> CR1_L=True && A2=None
ルール 8:CR1_L=True && B2=None && CR1_X=B && CR1_Y=_2 -> CR1_L=False && B2=Loading
ルール 9:CR1_L=False && B2=Loading && CR1_X=B && CR1_Y=_2 -> CR1_L=True && B2=None
ルール 10:CR1_Y=_2 -> CR1_Y=_1
ルール 11:CR1_X=A && CR2_X=C -> CR1_X=B
ルール 12:CR1_X=B -> CR1_X=A
ルール 13:CR2_L=True && B1=None && CR2_X=B && CR2_Y=_1 -> CR2_L=False && B1=Loading
ルール 14:CR2_L=False && B1=Loading && CR2_X=B && CR2_Y=_1 -> CR2_L=True && B1=None
ルール 15:CR2_L=True && C1=None && CR2_X=C && CR2_Y=_1 -> CR2_L=False && C1=Loading
ルール 16:CR2_L=False && C1=Loading && CR2_X=C && CR2_Y=_1 -> CR2_L=True && C1=None
ルール 17:CR2_Y=_1 -> CR2_Y=_2
ルール 18:CR2_L=True && B2=None && CR2_X=B && CR2_Y=_2 -> CR2_L=False && B2=Loading
ルール 19:CR2_L=False && B2=Loading && CR2_X=B && CR2_Y=_2 -> CR2_L=True && B2=None
ルール 20:CR2_L=True && C2=None && CR2_X=C && CR2_Y=_2 -> CR2_L=False && C2=Loading
ルール 21:CR2_L=False && C2=Loading && CR2_X=C && CR2_Y=_2 -> CR2_L=True && C2=None
ルール 22:CR2_Y=_2 -> CR2_Y=_1
ルール 23:CR2_X=B -> CR2_X=C
ルール 24:CR2_X=C && CR1_X=A -> CR2_X=B
```

図 9 制御記述
Fig. 9 Control rules

```
---検証式---
1,1,1,1,1,1,0,0,0,0,0,1
Next
not(val7="0")
---結果---
start(1,1,1,1,1,1,0,0,0,0,0,1)
-[0]->
goal(0,1,1,1,1,1,1,0,0,0,0,0,1)
```

図 10 時相論理 Next の実験
Fig. 10 Experiment of linear temporal logic Next

```
---検証式---
1,1,1,1,1,1,0,0,0,0,1,0
Finally
(val6="1" and val7="0" and val8="1")
             or (val9="1" and val10="0" and val11="1")
---結果---
start(1,1,1,1,1,1,0,0,0,0,0,1)
 ->[0]->[2]->[0]->[5]->
goal(0,1,1,1,1,1,1,0,0,0,0,0,1)
```

図 11 時相論理 Finally の実験
Fig. 11 Experiment of linear temporal logic Finally

```
---検証式---
0,0,0,0,0,0,1,0,0,1,1,0
Globally
val0="0" and val1="0" and val2="0" and val3="0"
   and val4="0" and val5="0" and val6="1" and val9="1"
---結果---
start(1,1,1,1,1,1,0,0,0,0,0,1)
 ->[0]->[2]->[0]->[5]->
goal(0,1,1,1,1,1,1,0,0,0,0,0,1)
```

図 12 時相論理 Globally の実験
Fig. 12 Experiment of linear temporal logic Globally

態空間への問い合わせでは 3 階 5 列のモデルに対しては非常に多くの時間がかかり，これ以降のサイズに対する検証は現実的な時間で検証が終了しないと判断できる．3 階 5 列のモデルにおいては 3 階 4 列のモデルに比べ，探索対象となる部分状態の数が 10 倍以上に拡大している．それらはすべて Neo4j に格納されているが，この探索アルゴリズムにおいて実行時間のボトルネックは DB への問い合わせである．次状態の探索を 3 階 5 列のモデルから生成された部分状態集合に対して実行する場合、Neo4j の応答時間が 5 秒ほど必要であった．3 階 4 列に対する問い合わせでは Neo4j の応答時間が 0.5 秒ほど必要であったことを考えると，検査不能になるのは当然であると考えられる．Neo4j の問い合わせ効率のために，各部分モデルを再分割することで，状態空間をさらに小さくする手法や，強連結成分の検出を行い，探索対象となる状態数の削減．また，Neo4j を使用した状態探索として，状態遷移 1 回ごとの状態の変化を複数回に渡り問い合わせる手法ではなく，複数回に渡る状態遷移を 1 度に問い合わせることで Neo4j への問い合わせ回数を減らし，処理時間の改善が見込める．

5. おわりに

5.1 まとめ

本研究では．この先行研究を用いて行うモデル検査において検証可能な条件を拡張した．拡張する検証条件として，先行研究でも用いられているモデル検査器 SPIN が使用している線形時相論理 (LTL) で記述される検証条件と同等な検証を行えるようにすることを目的とした．本研究では，LTL の時相論理のうち単項式を 1 つ使用して記述できる検証式を Neo4j への問い合わせによる状態探索によって検証する手法を提案した．複合した全体状態が検証条件を満たしているかを検査するために，全体状態を Neo4j に登録し再度問い合わせを行うことで部分的な条件の指定などの一般的な論理式を用いた複雑な条件を指定した検証が可能になった．また，時相論理 Globally での検証において，検証条件を各部分モデルへの問い合わせに適した内容になるように分割することで各部分状態への問い合わせのみで Globally の条件に違反する状態遷移を探索し，適用実験として，立体倉庫におけるクレーンと積荷の動作を模したモデルに対し検証を行い，提案手法による検証が行えることを示した．

5.2 今後の課題

今後の研究課題として以下の点が挙げられる．

5.2.1 対応する検査条件の拡張

本研究では LTL で使用される時相論理のうち，単項式を 1 つだけ使用した場合の検査条件の探索方法を提案した．本手法の有効性向上のため，今後 LTL の二項演算を用いた検査条件を検証するための問い合わせや複数の様相作用素を用いた検査式に対応することが求められる．

5.2.2 状態探索の高速化

各部分状態の数が多くなるにつれ，次状態を探索する処理時間が大幅に増加した．検証時間の増加を抑えるためにモデルをさらに分割する手法などが考えられるが，遷移 1 回ごとに状態を問い合わせる手法は Neo4j の持つ探索アルゴリズムを効果的に適用できているとは言えない．そこで，遷移 1 回ごとに状態を問い合わせずに検証条件を満たす部分状態に至るまでの複数の遷移をまとめて問い合わせる方法が考えられる．

参考文献

[1] 小山恭平, 小飼敬, 上田賀一, 山形知行, 武澤隆之, 中野利彦:情報制御システムのモ デル検査における状態空間分割による探索手法の提案, ソフトウェア工学の基礎 XIX, 近代科学社, Vol.38, pp.39-44, 2012.
[2] 宮島卓巳, 小飼敬, 上田賀一, 山形知行, 武澤隆之: モ ジュラ化手法によるモデル検査の検討とモジュラ検証の実用化, 電子情報通信学会技術研究報告. KBSE, 知能ソフトウェア工学, Vol. 114, No. 501(2015), pp. 25-30.
[3] 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,Vol33(2016),pp453-460.
[4] Amir Pnueli. The temporal logic of programs. In Proceeding of the 18th IEEE Sym - posium on Foundations of Computer Science, pp. 46-77, 1977.
[5] MySQL Community Server, http://www.mysql.com.
[6] Neo4j: The World's Leading Graph Database, https://neo4j.com/
[7] Holzmann, G. J., The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley, 2004 年. ISBN 0-321-22862-6.