グラフデータベースを用いたモデル検査の実装と評価

柳生 拓馬　安藤 崇央

近年, ソフトウェアの信頼性確保のために形式的検証手法が用いられている. 形式的検証手法の 1 つであるモデル検査は検査対象のシステムをモデル化し, そのシステムに対して調べたい性質の正当性を証明することができる. 本研究では, モデル検査の際にシステムモデルの役割を担う状態遷移図がグラフであることに着目し, 新たにグラフ情報をグラフの形のまま取り扱うグラフデータベースを用いたモデル検査器を実装した. 実装したモデル検査器に対して, 従来のモデル検査と同様の検査が行えるか検証し, 評価を行った.

1　はじめに

近年, ソフトウェアで制御される機器やサービスが社会に浸透するにつれ, そのシステム設計の正しさを保証することが重要になっている. システム設計が誤っていた場合, そのソフトウェアの不具合による影響が社会に大きな影響を与える可能性があるためである. そのような影響を与えないために, ソフトウェアの信頼性・安全性を確保する手段としてモデル検査が用いられる.

モデル検査とは, 形式的検証手法の 1 つである. モデル検査では, 検査対象のシステムの振る舞いを記述した状態遷移図と呼ばれる状態遷移モデルに対して, CTL [1] や LTL などで記述した論理検査式を施し, モデルに発生し得る全ての状態について網羅的に探索することで調べたい性質の正しさを保証している.

本研究では, モデル検査で検査対象のシステムの振る舞いを記述する際に使用される状態遷移図がグラフであることに着目し, グラフ情報をグラフのまま取り扱うデータベースであるグラフデータベース [2] を用いたモデル検査の手法について提案する.

グラフデータベースはノードとリレーション, そしてノードとリレーションのプロパティを定めることができるため, 状態遷移図の状態をノード, 状態遷移をリレーション, 詳細情報をプロパティで表現できる. これらの対応を用いて, グラフデータベース上に状態遷移図を表すグラフ情報を格納し, これをモデル検査におけるモデルとして使用する. そして, グラフデータベースを用いたモデル検査を行い, モデル検査におけるグラフデータベースの有用性を評価する.

2　関連研究

本研究の関連研究 [3] では, モデル検査器 SPIN [4] 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目的として検証を行い, LTL で使用される時相論理のうち, 単項式を 1 つだけ使用して記述できる検証式を Neo4j [5] への問い合わせによる状態探索によって検証できることを示している.

関連研究と本研究では, LTL ではなく CTL を用いる点とモデル検査におけるグラフデータベースの有用性を評価することを目的としている点で異なる.

Implementation and Evaluation of Model Checking Using Graph Databases.

Takuma Yagyu, 群馬大学大学院情報学研究科, Graduate School of Informatics, Gunma University.

Takahiro Ando, 群馬大学大学院情報学研究科, Faculty of Informatics, Gunma University.

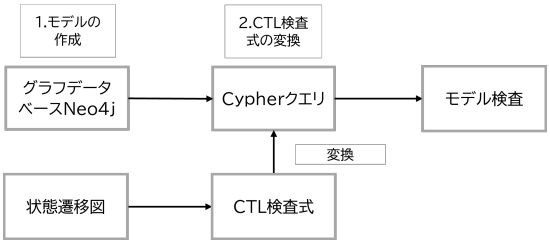

図 1　システムの構成

3　システム概要

3.1　システム構成

システムの構成を図 1 に示す. 検査対象のシステムの振る舞いをグラフデータベース Neo4j 上でモデル化し, CTL 検査式を本研究で作成した CTL 変換手順に基づいて Neo4j で用いる Cypher クエリに変換する. 変換したクエリをモデルに適用することでモデル検査を行う.

3.2　検査方法

3.2.1　モデルの作成

グラフデータベース Neo4j 上でシステムの状態と状態遷移, 状態ごとのプロパティをまとめたグラフをモデルとして作成する. グラフデータベースではノードやリレーションにプロパティを付けることが可能だが, 本研究ではプロパティをノードとして作成し, 状態にプロパティの関係でリレーションを繋げている. こうした理由としては, プロパティをノードとして作成することで, プロパティを持つときは状態からプロパティにリレーションを繋ぐことでプロパティを持っているということを表現できる. また, 同じプロパティを持つ状態があれば同じプロパティにリレーションを繋げればよいだけなので, プロパティノード 1 つだけで十分になる. これは, グラフの作成・探索を容易にする. また, プロパティが可視化されるため, どの状態がどのプロパティを持っているかが視覚的に理解できるためである.

3.2.2　CLT 式の変換

モデル検査における検査式である CTL 論理式を部分ごとに分解して, グラフデータベースでのクエリに変換する手順を以下に示す.

E($\phi \wedge \psi$) の変換手順

E($\phi \wedge \psi$) の意味：ϕ かつ ψ を満たす状態が存在する.

E($\phi \wedge \psi$) の変換手順を以下に示す.

1. ϕ と ψ をプロパティとして持つ状態ノードを探索する.ϕ かつ ψ を満たす状態が存在すれば, E($\phi \wedge \psi$) が成り立つことがいえる.

1 を変換したものを Cypher クエリ 1 に示す.

Cypher クエリ 1.　AND クエリ

```
MATCH (n:state)
WHERE (n)-[:property]->(:property{name:"ϕ"})
      AND (n)-[:property]->(:property{name:
      "ψ"})
RETURN n
```

このクエリは、モデル上のすべての状態ノードに対して, ϕ と ψ をプロパティノードとして持つかどうかを探索し, 一致する状態ノードがあればその状態ノードを返すというクエリである.

AXϕ の変換手順

AXϕ の意味：すべてのパス上のすべての状態において, 次の状態で ϕ が成り立つ.

AXϕ の変換手順を以下に示す.

1. すべての状態に対して, その状態と 1 つ前の状態ノードを探索する.
2. その状態が ϕ をプロパティノードとして持つなら 1 を返し, それ以外は 0 を返す. すべての状態で 1 を返さない場合は AXϕ は成り立たないといえる.

1, 2 をまとめて変換したものを Cypher クエリ 2 に示す.

Cypher クエリ 2.　AX クエリ

```
MATCH (n:state)-[:NEXT]->(m:state)-[:property]
      ->(a:property)
RETURN
CASE a.name
 WHEN "ϕ" THEN 1
 ELSE 0
END AS result,n.name
```

このクエリは, すべての状態に対して, その状態と

1 つ前の状態ノードを探索し, その状態が ϕ をプロパティノードとして持つなら 1 を返し, それ以外は 0 を返すクエリである.

EXϕ の変換手順

EXϕ の意味：次の状態遷移で ϕ が成り立つ状態へ遷移するパスが存在する.

EXϕ の変換手順を以下に示す。

1. ϕ をプロパティノードとして持つ状態ノードを探索する.
2. その状態の 1 つ前の状態ノードを探索する. ϕ が成り立つ状態ノードへ遷移するある状態ノードが存在すれば, EXϕ が成り立つことがいえる.

1, 2 をまとめて変換したものを Cypher クエリ 3 に示す.

Cypher クエリ 3.　EX クエリ

```
MATCH (n:state)-[r:NEXT]->(m:state)
WHERE (m)-[:property]->({name:"ϕ"})
RETURN n,m,r
```

このクエリは, ϕ をプロパティノードとして持つ状態ノードとその状態ノードに遷移する 1 つ前の状態ノードを状態遷移もふまえて返すクエリである.

AGϕ の変換手順

AGϕ の意味：パス上のすべての状態に対して常に ϕ が成り立つ.

AGϕ の変換手順を以下に示す.

1. ϕ をプロパティノードとして持つ状態ノードを探索し, ϕ をプロパティノードとして持つ状態数をカウントする.
2. モデルの状態数と ϕ をプロパティノードとして持つ状態数が等しければ, AGϕ が成り立つことがいえる.

1 を変換したものを Cypher クエリ 4 に示す. 2 は Neo4j 上でモデル全体の状態数が確認できるので, それと比較すればパス上のすべての状態に対して常に ϕ が成り立つかどうかがわかる.

Cypher クエリ 4.　AG クエリ

```
MATCH (n:state)
WHERE (n:state)-[:property]->(:property{name:
      "ϕ"})
RETURN count(n)
```

このクエリは, ϕ をプロパティノードとして持つ状態を探索し, ϕ をプロパティノードとして持つ状態数をカウントして返すクエリである.

EGϕ の変換手順

EGϕ の意味：常に ϕ が成り立つパスが存在する.

EGϕ の変換手順を以下に示す.

1. ϕ をプロパティノードとして持つ状態ノードと状態遷移を探索する.
2. 常に ϕ が成り立つ状態遷移パスがあれば EGϕ が成り立つことがいえる.

1 を変換したものを Cypher クエリ 5 に示す. 2 は探索した状態遷移パスを見れば, ϕ が常に成り立つパスが存在するか確認できる.

Cypher クエリ 5.　EG クエリ

```
MATCH (n:state)
WHERE (n:state)-[:property]->(:property{name:
      "ϕ"})
RETURN *
```

このクエリは, ϕ をプロパティノードとして持つ状態と状態遷移を返すクエリである.

AFϕ の変換手順

AFϕ の意味：すべてのパス上のすべての状態において, いつかある状態で ϕ となることが成り立つ.

AFϕ の変換手順を以下に示す.

1. ϕ をプロパティノードとして持つ状態ノードを探索する.
2. すべての状態ノードからその状態ノードのどれかへ到達するパスを探索する.
3. 1 つでも到達する状態パスがなければ, AFϕ は成り立たないといえる.

1, 2 をまとめて変換したものは Cypher クエリ 6 と同じものとなる. このクエリにより, すべての状態ノードからその状態ノードのどれかへ到達するパスを探索できているので, 3 はクエリの結果から確認することで証明できる.

EFϕ の変換手順

EFϕ の意味：いつか ϕ が成り立つある状態に到達するパスが存在する.

EFϕ の変換手順を以下に示す.

1. ϕ をプロパティノードとして持つ状態ノードを探索する.
2. ある状態ノードからその状態ノードに到達する

パスが存在するか探索する．到達する状態パスが存在すれば，EFϕ が成り立つことがいえる．

1, 2 をまとめて変換したものを Cypher クエリ 6 に示す．

Cypher クエリ 6.　EF クエリ

```
MATCH (n:state)
WHERE (n)-[:property]->({name:"φ"})
WITH n
MATCH (a:state)-[*]->(n)
RETURN *
LIMIT 3000
```

このクエリは，ϕ をプロパティノードとして持つ状態ノードを探索し，その状態ノードへ到達するパスを持つ状態ノードとパスを返すクエリである．LIMIT の部分は結果の出力行数を制限する部分なので，状態数により変化する．

A(ϕUψ) の変換手順

A(ϕUψ) の意味：すべてのパス上のすべての状態において，ψ が成り立つまで ϕ が成り立つ．

A(ϕUψ) の変換手順を以下に示す．

1. ψ をプロパティノードとして持つ状態ノードを探索する．
2. ϕ をプロパティノードとして持つ状態ノードから ψ をプロパティノードとして持つ状態ノードどれかに到達するパスがあるか探索する．
3. そのパス上で 1 つでも ϕ が成り立たない状態ノードがあれば，A(ϕUψ) は成り立たないといえる．

1, 2 をまとめて変換したものは Cypher クエリ 7 と同じものとなる．このクエリは，ψ をプロパティノードとして持つ状態ノードを探索し，その状態ノードへ到達するパスで ϕ をプロパティノードとして持っているパスを返すクエリなので，3 はクエリの結果を確認することで証明できる．

E(ϕUψ) の変換手順

E(ϕUψ) の意味：ψ が成り立つまで ϕ が成り立つパスが存在する．

E(ϕUψ) の変換手順を以下に示す．

1. ψ をプロパティノードとして持つ状態ノードを探索する．
2. ある状態ノードからその状態ノードに到達するパスが存在するか探索する．
3. 探索したパスに対して，ψ をプロパティノードとして持つ状態ノードに到達するまでの状態ノードすべてでプロパティノード ϕ を持っているか探索する．

1, 2, 3 をまとめて変換したものを Cypher クエリ 7 に示す．

Cypher クエリ 7.　EU クエリ

```
MATCH (n:state)
WHERE (n)-[:property]->({name:"ψ"})
WITH n
MATCH (m)-[r]->(n)
MATCH (m)-[:property]->(:property{name:"φ"})
RETURN *
LIMIT 3000
```

このクエリは，ψ をプロパティとして持つ状態ノードを探索し，その状態へ到達するパスで ϕ をプロパティとして持っているパスを返すクエリである．LIMIT の部分は結果の出力行数を制限する部分なので，状態数により変化する．

3.2.3　モデル検査

節 3.2.1 で作成したシステムのモデルに対して，検証したいシステムの性質を節 3.2.2 に従って CTL 式から変換した Cypher クエリで投げることにより，実行結果として，従来のモデル検査における状態遷移図に対して CTL 式を実行した時と同様の結果が得られる．

4　システムの実行例

実行例として，[6] から引用した以下の例題を題材に構築したシステムでモデル検査を実施する．

> 窓辺に 1 列，8 個の花が並んでいる．花たちはそれぞれ 1 日単位で開いたり閉じたりする．ある花が明日開くかどうかは，ご近所（その花とその花の両端の花を合わせた 3 個，端の場合は 2 個）のうち，いくつの花が開いているか（混み具合）で変化する．各花は明日の状態を次のように決める．
>
> 今日のご近所の混み具合が 0 か 3 なら，明日は閉じている．今日のご近所の混み具合が 1 なら，明日は開いている．今日のご近所の混み具合が 2 なら，明日は開いているかも

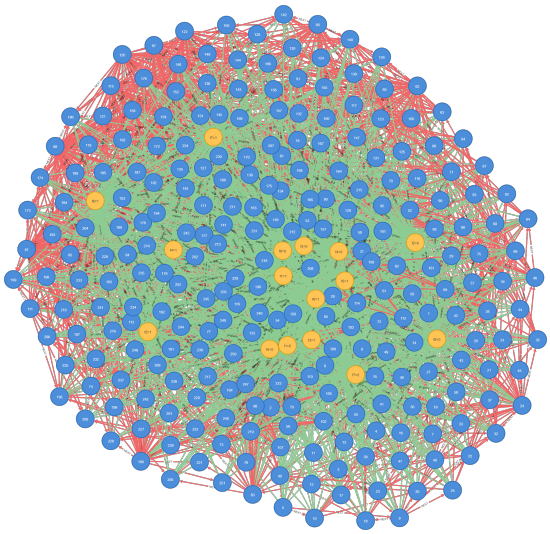

図 2　flowers モデル

しれないし, 閉じているかもしれない.

さて, 今日は左端の花 1 個だけが開いているとする. この花たちが 8 個すべて開花する日はあるのでしょうか？

4.1　実行結果

実行は節 3.2 の検査方法の手順に従って行う. まずは, システムのモデルをグラフデータベース Neo4j 上で作成する.

グラフデータベース Neo4j 上で作成したシステムのモデルを図 2 に示す.

このモデルは、状態を表すノード（state[青]）と状態遷移を表すリレーション（NEXT[赤]）, そして状態のプロパティを表すノード（property[黄]）と状態がプロパティを持つことを示すためのリレーション（property[緑]）で構成されている。状態と状態遷移, プロパティはクエリで入力して作成していき, モデルを完成させた.

次に, 作成したモデルに対して検査式を適用するために CTL 式を Cypher クエリに変換する. 今回のシステムの終了状態はすべての花が開花している状態なので, 終了状態へのパスを得る CTL の検査式は次のようになる.

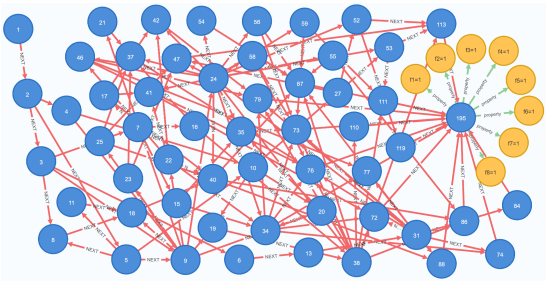

図 3　Cypher クエリ 8 を実行した結果

$$AG!(f1 = 1 \wedge f2 = 1 \wedge f3 = 1 \wedge f4 = 1 \wedge f5 = 1 \wedge f6 = 1 \wedge f7 = 1 \wedge f8 = 1)$$

この検査式を CTL 変換手順 3.2.2 に基づいて変換する.

変換したクエリをクエリ 8 に示し, クエリを実行した結果を図 3 に示す.

Cypher クエリ 8.　終了状態へ到達するパスを返すクエリ

```
MATCH (n:state{name:"1"})-[r:NEXT*]->(m:state),
      (m)-[:property]->(a)
WHERE (m)-[:property]->({name:"f1=1"}) AND
      (m)-[:property]->({name:"f2=1"}) AND
      (m)-[:property]->({name:"f3=1"}) AND
      (m)-[:property]->({name:"f4=1"}) AND
      (m)-[:property]->({name:"f5=1"}) AND
      (m)-[:property]->({name:"f6=1"}) AND
      (m)-[:property]->({name:"f7=1"}) AND
      (m)-[:property]->({name:"f8=1"})
RETURN *
```

図 3 に対して, 終了状態へ到達する最短パスをグラフ DB を用いたモデル検査の実行結果として図 4 に示す.

この実行結果をモデル検査器 NuSMV [7] での実行結果と比較する. グラフ DB で表した NuSMV での実行結果を図 5 に示す.

図 4 と図 5 を比較すると, 終了状態へ到達するステップ数は同じだが, 状態遷移の仕方は異なる遷移の仕方を示した. この比較結果は, 図 3 より, 同じステップ数で終了状態へ到達するパスが複数存在していたことから妥当である. また, 図 5 の状態遷移パスは, 図 3 に示した初期状態から終了状態へ遷移するパ

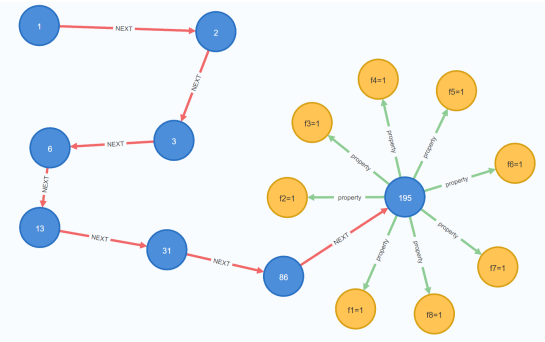

図 4　グラフ DB を用いたモデル検査の実行結果

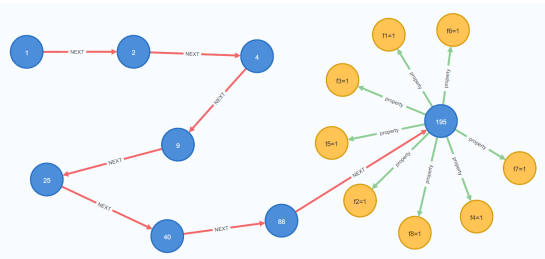

図 5　モデル検査器 NuSMV での実行結果

ス上に存在したため, 従来のモデル検査器と同様の検査結果を得られることが確認できた.

比較結果から新たに確認できたこととして, 従来のモデル検査器である NuSMV では, 初期状態から終了状態へ遷移するパスは 1 つしか得られなかったが, グラフデータベースを用いたモデル検査では, 初期状態から終了状態へ遷移するパスがすべて得られることが確認できた.

5　システムの評価

本研究で実装したシステムの評価を行う.

5.1　CTL 式変換手順

モデル検査に使用される CTL 検査式をグラフデータベース Neo4j で作成したモデルに適用するために, CTL 検査式を Cypher クエリに変換する手順を本研究で提案した.

本研究で作成した CTL 式変換手順は十分な実行結果を得られるクエリが作成できることが確認できた.

新たな発見として, CTL 式変換手順に従って変換したクエリのいくつかは, A [All] と E [Exists] 関係なく, 同じクエリで両方を証明できるものもあった. この特徴は, 従来のモデル検査における検査式と異なる部分である.

5.2　グラフデータベースを用いたモデル検査

グラフデータベース Neo4j 上でシステムのモデルを作成し, 検査式である CTL 式を Neo4j 上で用いる Cypher クエリに変換して, 変換したクエリを作成したモデルに適用した.

実行結果を従来のモデル検査器の実行結果と比較した際に同様の実行結果が得られたことから, グラフデータベースを用いたモデル検査の実装はできたといえる. しかし, モデル検査を行うまでにまだまだ改善できる部分が多い. 特に, モデルの作成では, 検査対象のシステムの状態や状態遷移をすべてグラフデータベースに手作業で入力していたため多くの時間を要した.

改善策として, 自動化されていない工程をプログラムで自動化する作業を現在行っている.

5.3　モデル検査におけるグラフデータベースの有用性

本研究の目的である「モデル検査におけるグラフデータベースの有用性」を検証結果から評価する.

本研究の結果として, グラフデータベースを用いたモデル検査の実行結果と従来のモデル検査器である NuSMV での実行結果を比較した際に同様の結果が得られる点や従来のモデル検査器と異なり, 検査式を満たすすべての状態遷移パスを見やすくグラフとして出力できる利点があることが確認できた.

現状の課題・懸念点としては、膨大な状態数を持つシステムを扱えるかどうかの懸念点とシステムの状態と状態遷移をグラフデータベース上に入力するのに時間を要するという課題がある.

以上より, グラフデータベースのモデル検査における有用性については, モデル検査に使用する利点はあ

るが, 使用するには利用環境の整備が必要であると評価する.

6 まとめ

本研究の結果として, グラフデータベースを用いたモデル検査は従来のモデル検査器 NuSMV と同様の結果が得られる点や従来のモデル検査器と異なり, 検査式を満たすすべての状態遷移パスを見やすくグラフとして出力できる利点があることが確認できた. しかし, 現時点では, 本システムを用いたモデル検査の実施にはまだ多くの自動化されていない工程があるため、利用環境を整えることが必要である. 今後の展望としては, グラフデータベースを用いたモデル検査の実施にはまだ多くの自動化されていない工程があるため, グラフデータベースを用いるための利用環境の整備を引き続き行っていきたいと考えている. また, モデル検査で問題となっている状態爆発問題に対する対策を本研究の提案手法であるグラフデータベースを用いたモデル検査で行えるのではないかと考えているため, 従来のモデル検査器では状態爆発を引き起こすような膨大な状態数を持つシステムを本手法で扱えるか検証し, グラフデータベースのさらなる有用性を示していきたいと考えている.

参 考 文 献

[1] 鹿島亮. コンピュータサイエンスにおける様相論理. 森北出版株式会社, 2022.

[2] Oracle. グラフ・データベースとは—Oracle 日本. https://www.oracle.com/jp/autonomous-database/what-is-graph-database/, 2024.

[3] 久野和敏, 上田賀一, 小飼敬. グラフデータベースを用いたモデル検査手法の提案. 研究報告ソフトウェア工学 (SE) Vol. 2018-SE-198, No. 9, pp. 1–8, March 2018.

[4] 中島震. SPIN モデル検査. 近代科学者, 2008.

[5] Inc. Neo4j. Neo4j Graph Database & Analytics | Graph Database Management System. https://neo4j.com/, 2024.

[6] 産業技術総合研究所システム検証研究センター. 4 日で学ぶモデル検査 初級編. 壮光舎印刷株式会社, 2006.

[7] Shin Saito. NuSMV によるモデル検査入門 (1) ステートマシンを定義する. https://qiita.com/shinsa82/items/cd4d95c616bf1da852ce, 2020.