Vol.2021-SE-209 No.6
2021/11/25

グラフクエリに対するミューテーションテストの提案

有若新悟[1]　土屋 達弘[1]

概要：本研究では，グラフデータベースに対するクエリであるグラフクエリを対象に，ミューテーションテストを実行する手法を提案する．具体的には，まずバグを模倣するミューテーションオペレータを複数提案する．その上で，クエリをパースして AST を構築し，提案したミューテーションオペレータを適用して AST を変化させるツールを開発する．

On mutation testing for graph queries

1. はじめに

ミューテーションテスト (mutation testing) では，テスト対象のプログラムを改変し，人為的に不具合を埋め込んだミュータントと呼ばれるプログラムを生成する [1], [2]．このようなミュータントを用いることで，たとえば，テストの実行が十分かを評価することが可能となる．この場合，元のプログラムに対して実行したテストをミュータントに適用することで，ミュータントのバグを正しく検出できるかを評価する．バグが検出できないようなミュータントが多ければ，テストが不十分であることが推測できる．

ミューテーションテストにおける典型的なバグとしては，分岐条件における演算子の改変等が挙げられる．しかし，プログラムの内部からデータベースへのアクセスを行っている場合，このようなプログラムの書き換えでは，データベースへのアクセスを担うクエリのテストに利用することはできない．そこで本研究では，バックエンドとしてグラフデータベースを用いるプログラムを想定して，グラフデータベース用のクエリであるグラフクエリに対するミューテーションテストについて議論する．

より具体的には，グラフクエリを記述するクエリ言語として Cypher を取り上げ，Cypher で記述されたクエリが与えられたときに，機械的にバグを埋め込むことで，ミュータントとなる Cypher クエリを生成する方法を提案する．そのために，まず，Cypher クエリに対してバグを埋め込むミューテーションオペレータを 5 つのクラスに分けて定義する．その上で，クエリに定義したミューテーションオペレータを適用し，実際にミュータントを生成する手法について述べる．

データベースのクエリに対するミューテーションテストに関しては，これまで SQL クエリに対する研究が知られている [2], [3]．SQL クエリを用いる関係データベースは，もっとも普及しているデータベースであるが，近年，これに代わって，いわゆる NoSQL データベースと呼ばれる新しい種類のデータベースも浸透しつつある．グラフデータベースは NoSQL の一種であり，ネットワーク構造を基本としたデータ表現を採用している．そのため，ネットワーク上のデータの扱いに優れており，その代表的なクエリ言語である Cypher は，頂点や辺といったネットワークの形状を視覚的に分かりやすく表現できるなど，SQL とは異なる性質を持つ [4]．提案するミューテーションオペレータは，Cypher の独自の特性を考慮して設計した．我々の知る範囲では，グラフクエリに対するミューテーションテストの研究はこれまで行われておらず，本研究が最初のものである．

以降，本稿では，まず 2 節でグラフデータベース，および，グラフクエリについて簡単に述べる．3 節では，Cypher クエリからミュータントを生成する過程を例を用いて説明する．4 節では，設計した 5 クラスのミューテーションオペレータについて概説する．5 節では，現在制作中のミュータントを生成するツールについて簡単に触れる．6 節でまとめと今後の研究の方向について述べる．

[1] 大阪大学
Yamadaoka, Suita-shi, Osaka-fu 565–0871, Japan

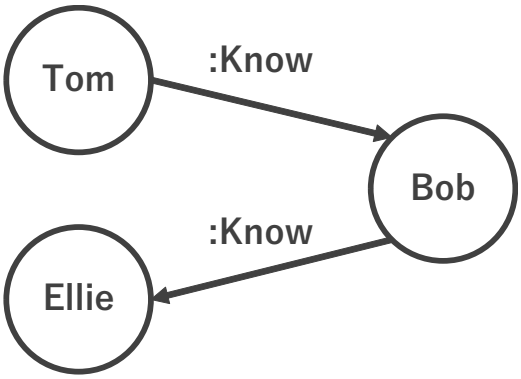

図 1 人間関係を模したグラフ構造データの例

2. グラフデータベースとグラフクエリ

2.1 グラフデータベース

グラフデータベースは，グラフ構造を用いてデータを管理するデータベースである [5]．グラフデータベースは，データの構造や問い合わせの方法が従来の関係データベースとは大きく異なることから，NoSQL とも呼ばれている．グラフデータベースが扱うグラフ構造のデータとは，グラフ理論における，ノード（頂点）とノード同士を繋ぐ有向のリレーションシップ（辺）によって構成されるデータである．グラフデータベースは，これらのノードやリレーションシップに対して，プロパティやラベルと呼ばれるデータを付与することで，容易に検索が可能なグラフ構造のデータベースを構成している．

例として，図 1 に，人間関係を模した簡単なグラフ構造のデータを示す．このデータは，ノードひとつひとつが人物を表しており，人物の名前は，それぞれのノードの name プロパティに格納されている．また，ノード同士を繋ぐ有向のリレーションシップは，人物同士の関係性を示している．ここでは，リレーションシップに Know というラベルが付与されており，どの人物が誰のことを知っているかをリレーションシップによって表現している．

2.2 グラフクエリ

グラフデータベースに問い合わせを行うクエリをグラフクエリと呼ぶ．Cypher は，グラフクエリを記述するための代表的なクエリ言語の 1 つである．Cypher の構文の多くは SQL に類似しているが，SQL にはない特徴的な構文として，パターンマッチ構文を持つ．パターンマッチ構文は，ノードを丸括弧，リレーションシップを矢印と角括弧で表し，グラフ構造のパターンを記述することで，記述したパターンに一致するデータの検索を行う．例として，パターンマッチ構文を用いたグラフクエリを以下に示す．

```
1  MATCH (a)-[:Know]->(b)
2  WHERE b.name = “Bob”
3  RETURN a.name
```

このクエリは，図 1 のグラフ構造データに対するクエリであり，「Bob のことを知っている人の一覧」を取得している．パターンマッチ構文は，クエリの一行目で MATCH キーワードと共に使用されている．Bob のことを知っている人物は Tom のみであるため，このクエリの実行結果は Tom となる．

3. グラフクエリに対するミュータントの生成

グラフクエリに対するミューテーションテストでは，クエリの一部を機械的に書き換えることで，意図的に不具合を埋め込んだ，ミュータントと呼ばれるクエリを生成する必要がある．例えば，2 節で説明したグラフクエリに対して，「パターンマッチ構文におけるリレーションシップの向きを変更する」という操作を行うと，次の 2 つのミュータントが生成できる．

```
1  MATCH (a)<-[:Know]-(b)
2  WHERE b.name = “Bob”
3  RETURN a.name
```

```
1  MATCH (a)-[:Know]-(b)
2  WHERE b.name = “Bob”
3  RETURN a.name
```

1 つ目のミュータントは，リレーションシップの向きを，右向きから左向きに変更したものである．この場合，クエリの意味は「Bob が知っている人の一覧」となり，実行結果は Ellie になる．

2 つ目のミュータントは，リレーションシップの方向指定を無くしたものである．この場合，リレーションシップの向きに関係なくパターンマッチングが行われる．そのため，クエリの意味は「Bob のことを知っている人，または Bob が知っている人の一覧」となり，実行結果は Tom と Ellie になる．

このように，クエリの一部を機械的に書き換えることで生成されたミュータントは，通常，クエリの動作になんらかの変化が起きるため，適切な入力（データベースの状態）を与えることで，本来期待される出力とは異なる出力を得ることができる．しかし，場合によっては，クエリの動作が全く変化せず，どんな入力を与えても，元のクエリと同じ出力になることがある．このようなクエリは，等価ミュータントと呼ばれ，極力生成されないように工夫をすることが望ましい．

4. ミューテーションオペレータ

本節では，Cypher で記述されたクエリに不具合を埋め込む操作の方法を定義するミューテーションオペレータについて，我々が設計した 5 つのクラスを紹介する．これは，我々が定義した 27 個のミューテーションオペレータを，その意味や特徴に応じて大きく 5 つのクラスに分類したものである．なお，本研究では現時点において，グラフデータベースからデータを検索するクエリのみを対象として，ミューテーションオペレータの設計を行なっている．

4.1 パターンマッチに対する操作

Cypher において最も重要かつ特徴的な要素は，パターンマッチ構文によるグラフ構造のデータの検索が行える点である．この構文は，直感的にグラフ構造のパターンを記述することができ，グラフ構造データの扱いを容易にしている．しかし，このパターンマッチ構文は，他のプログラミング言語，及びクエリ言語にはない独特な機能である．そのため，クエリの開発者が十分に Cypher に習熟していない場合，パターンの記述方法を誤ってしまうことが考えられる．このクラスのミューテーションオペレータは，そうしたパターンの記述に関連する不具合を模倣し，Cypher に特有の不具合を見つけるのに役立つ．

4.2 その他の構文に対する操作

ORDER BY 構文や UNION 構文など，Cypher には SQL と類似した構文がいくつか存在する．このクラスのミューテーションオペレータは，そういった構文を含む，パターンマッチ構文以外の構文に関連する不具合を模倣する．これらの構文には，一般のプログラミング言語とは異なる，データベースクエリに特有の機能が多く含まれる．

4.3 式の演算子に対する操作

このクラスのミューテーションオペレータは，主に WHERE 構文の内部の条件式に出現する，論理演算子，比較演算子，算術演算子などの演算子に関連する不具合を模倣する．これらのミューテーションオペレータは他のプログラミング言語のミューテーションオペレータと共通する部分も多いが，Cypher の構文に合わせて，一部改良，及び拡張を加えている．

4.4 NULL の扱いに対する操作

Cypher における論理体系は，SQL と同じく，3 値論理を採用している．Cypher における 3 値論理は，TRUE（真）と FALSE（偽）に加えて，NULL も許容する．この 3 値論理において，特に NULL を含む論理式の評価の挙動は，あまり馴染み深いものではなく，不具合を引き起こす要因になりやすい．また，論理式の評価以外においても，NULL の扱いは難しい．このクラスのミューテーションオペレータは，このような NULL の扱いに関連する不具合を模倣する．

また，テストケースとして実行されるクエリの出力には様々な値が含まれることが望ましい．すなわち，優れたテストケースには，実行結果に NULL が含まれるようなものが必要である．そのため，このクラスのミューテーションオペレータは，不具合を模倣するほかに，優れたテストケースを作成することを強制するためのミュータントも生成する．

4.5 識別子に対する操作

このクラスのミューテーションオペレータは，例えば，アクセスするべきプロパティと異なるプロパティにアクセスしてしまった場合のような，識別子に関連する不具合を模倣する．ただし，グラフデータベースは一般的にスキーマレスのデータベースであるため，クエリ内の識別子を，クエリには出現しない（がデータベースには存在する）識別子に置き換えることは想定していない．また，クエリに出現する識別子が持つ値のデータ型や，それが含まれる式の型を，静的な解析によって事前に確定させることは基本的にはできないため，単純に識別子を置き換えた場合，実行時に型エラーになる，無能なミュータント（incompetent mutant）が生成される可能性がある．この点について，文献 [6] では，実行時に得られる型情報を用いて適切なミュータントのみを実行するという動的なアプローチを用いることで，無能なミュータントを排除する方法が提案されている．

5. ツール実装

現在，入力された Cypher クエリに対して，4 節で説明したミューテーションオペレータを適用し，クエリのミュータントを生成するツールを開発している．このツールはまず，Cypher のパーサライブラリを用いて，入力されたクエリを AST（抽象構文木）に変換する．そして，木構造である AST の各ノードを順番に走査し，ミューテーションオペレータが適用できる箇所を見つけるたびに，AST を改変して AST のミュータントを生成する．最後に，AST のミュータントからクエリを再構築することで，クエリのミュータントを作成する．一旦 AST を経由することで，クエリの一部の情報（省略可能な括弧の有無など）が失われてしまう場合があるが，クエリ自体の意味や実行結果そのものに影響を及ぼさないようにクエリを再構築することができるため，ミューテーションテストの実施には一切影響を及ぼさない．

6. おわりに

本研究では，グラフデータベースに対して Cypher 言語で記述されたグラフクエリの，ミューテーションテストの方法を提案した．Cypher 言語の特性を考慮してミュータントを生成するミューテーションオペレータを 5 つのクラスに分けて設計した．また，これらのオペレータに基づいて与えられたグラフクエリを操作してミュータントを生成する方法について簡単に述べた．

今後は，提案したクラスに基づいてミューテーションオペレータの種類を拡充する．その上でミュータントを生成するツールを完成させるとともに，ツールを用いて生成したミュータントを用いて，プログラムのテスト状況を評価する枠組みを開発する．

謝辞

本研究は JSPS 科研費 JP20K11747 の助成を受けたものである．

参考文献

[1] Woodward, M.: Mutation testing—its origin and evolution, *Information and Software Technology*, Vol. 35, No. 3, pp. 163–169 (online), DOI: doi.org/10.1016/0950-5849(93)90053-6 (1993).
[2] Zhou, C. and Frankl, P.: Mutation Testing for Java Database Applications, *2009 International Conference on Software Testing Verification and Validation*, pp. 396–405 (online), DOI: 10.1109/ICST.2009.43 (2009).
[3] Tuya, J., Suárez-Cabal, M. J. and de la Riva, C.: Mutating database queries, *Information and Software Technology*, Vol. 49, No. 4, pp. 398–417 (online), DOI: 10.1016/j.infsof.2006.06.009 (2007).
[4] Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P. and Taylor, A.: Cypher: An Evolving Query Language for Property Graphs, *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, New York, NY, USA, Association for Computing Machinery, pp. 1433–1445 (online), DOI: 10.1145/3183713.3190657 (2018).
[5] Robinson, I., Webber, J. and Eifrem, E.: *Graph databases: new opportunities for connected data (2nd Edition)*, O'Reilly (2015).
[6] Hashish, N. A.: Mutation Analysis of Dynamically Typed Programs, PhD Thesis, University of Hull (2013).