DEIM Forum 2021 A13-2

Shape Expression Schemaにおけるパターン問合せの包含性判定

藤本　春奈†　　鈴木　伸崇††

† 筑波大学情報学群知識情報・図書館学類　〒 305–8550 茨城県つくば市春日 1-2
†† 筑波大学図書館情報メディア系　〒 305–8550 茨城県つくば市春日 1-2
E-mail: †s1711556@s.tsukuba.ac.jp, ††nsuzuki@slis.tsukuba.ac.jp

あらまし　本研究では，グラフデータのスキーマ言語である Shape Expression Schema(ShEx)，及びグラフデータに対する問合せのパターン問合せに着目する．包含性判定は問合せ最適化などに有用であるため，莫大なデータ量を持つことの多いグラフデータにとって重要である．一般に，スキーマが定義されていないグラフデータに対して行う通常のパターン問合せであれば，部分グラフ同型問題で包含性の有無を判定できる．しかし，スキーマの存在を仮定し，射影演算したパターン問合せの包含性を判定する場合においては，部分グラフ同型問題を解くだけでは判定できない包含性が存在する．そこで本稿では，射影演算した 2 つのパターン問合せと ShEx スキーマが与えられた際に，包含性の有無を判定する手法を提案する．

キーワード　Shape Expression Schema，パターン問合せ，包含性判定

1　はじめに

近年，グラフデータの普及が進んでいる．また，こうしたグラフデータに対する様々なスキーマ言語が提案されている．グラフデータは一般的に莫大なデータ量を持つ場合が多いため，より効率のよい問合せ処理が必要とされている．包含性判定は問合せ最適化などに有用であり，先行研究では XPath [1] や SPARQL [2] などの問合せ言語の包含性判定に関する研究がなされている．そのため，効率のよい問合せ処理が必要とされるグラフデータにおいて，包含性判定は重要な意味合いを持つ．

本研究では，RDF データ (グラフデータ) のスキーマ言語である Shape Expression Schema (以下 ShEx) [3]，及びグラフデータに対する問合せのパターン問合せに着目する．ShEx は，Regular Bag Expression という規則に基づいて各ノードに「型」を割り当てることが特徴のスキーマ言語である．グラフデータのスキーマは，ShEx の他にも RDF Schema [4] やグラフスキーマが提案されている．しかし，RDF Schema はオントロジ記述言語としての性質が強く，スキーマ言語としてのセマンティックスが W3C で定義されていない．また，グラフスキーマはスキーマをグラフとして表現するが，その表現力が低い．そこで，グラフデータの普及に伴い，より適切に記述できるスキーマが必要となり，ShEx が提案された経緯がある．

パターン問合せは，データグラフにおいて指定したパターングラフと同型の部分グラフを検索する問合せであり，グラフデータに対する問合せとして最も一般的な問合せである．本研究では，出力すべきノード変数を指定して取り出す射影演算を行った問合せを対象としている．

一般に，スキーマが定義されていないグラフデータに対して行う通常のパターン問合せであれば，部分グラフ同型問題で包含性の有無を判定できる．しかし，スキーマの存在を仮定し，射影演算したパターン問合せの包含性を判定する場合においては，部分グラフ同型問題を解くだけでは判定できない包含性が存在する．こうした ShEx スキーマを仮定し，射影演算したパターン問合せの包含性を判定するアルゴリズムは，著者の知る限り考案されていない．そこで本研究では，射影演算した 2 つのパターン問合せと ShEx スキーマが与えられた際に，包含性の有無を判定する手法を提案する．

提案するアルゴリズムの前提として，使用するパターン問合せの検索対象のデータグラフは使用する ShEx スキーマに対して妥当なデータグラフであると仮定し，パターン問合せは ShEx スキーマに対して充足可能であるとする．ここでの充足可能とは，問合せ q と ShEx スキーマ S に対して，S に妥当なデータで q の解が空でないものが存在することを意味する．まず提案アルゴリズムでは，包含性を判定する 2 つのパターン問合せにおいて，ノード数が 3 以上の 2 連結成分 (biconnected component) を含むかどうかを判定する．ここで，2 連結成分は無向グラフにおける概念であるが，本研究においては有向グラフにおいてエッジの向きを無視した際のノード間の繋がりに対してこの概念を用いる．パターン問合せのどちらかにノード数が 3 以上の 2 連結成分を含む場合は，その対応部分における包含性を確認した後に，次の処理に移る．これは，2 つのパターン問合せが同じ変数とラベルを使っていても，3 つ以上のノード同士が繋がっているものとそうでないものでは解が異なるために，行う必要がある処理である．次に，ShEx スキーマの定義に従い，一方のパターン問合せにエッジおよびノードを追加する。エッジを追加する際には包含性判定のための条件を設定しており、条件を満たすエッジのみ追加する。この条件は，エッジを追加することによって，パターン問合せの解が変わることのないようにするために設定している．こうして変更したパターン問合せともう一方のパターン問合せを用いて，一方が他方の部分グラフとなっているかを確認し，包含性を判定する。

以上の提案アルゴリズムを実装し，ShEx スキーマとその ShEx スキーマに対して充足可能なパターン問合せを用いて評

価実験を行った．その結果，提案アルゴリズムにより 2 つのパターン問合せにおける包含性判定を効率よく行うことができることを確認した．

本文の構成は次の通りである．第 2 章では，グラフ，パターン問合せ，ShEx，包含性判定の定義を述べる．第 3 章では，提案アルゴリズムについて述べる．第 4 章では，評価実験について述べる．第 5 章では，本研究のまとめと今後について述べる．

2 諸 定 義

本章では，本研究で扱うグラフ，パターン問合せ，Shape Expression Schema，充足可能性問題について述べる．

2.1 グ ラ フ

本研究ではラベル付き有効グラフ (以下グラフ) を対象とする．このグラフは, ラベル，ノード，エッジの 3 つの要素から構成される．Σ 上のグラフ G を，$G = (V, E)$ とする．ここで，Σ はラベルの集合，V はノードの集合，$E \subseteq V \times \Sigma \times V$ はラベル付き有向辺の集合である．例として，図 1 にグラフ G_0 を示す．

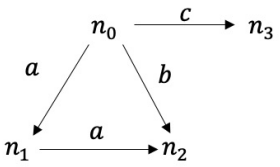

図 1　グラフ G_0

ここで，グラフ G_0 は，

$\Sigma = \{a, b, c\}$
$V = \{n_0, n_1, n_2, n_3\}$
$E = \{(n_0,\ a,\ n_1), (n_0,\ b,\ n_2), (n_0,\ c,\ n_3), (n_1,\ a,\ n_2)\}$

である. また，グラフ内にある辺において，辺を出力するノードを出力ノード，受け取るノードを入力ノードと呼称する．図 2.1 においてラベルが c であるエッジを例に挙げると，出力ノードは n_0，入力ノードは n_3 となる．

2.2 パターン問合せ

本節では，本研究で用いるパターン問合せについて定義する．

パターン問合せは，データグラフにおいて，指定したパターングラフと同型の部分グラフを検索する問合せである．本研究では，出力すべきノード変数を指定して取り出す射影演算を行った問合せを対象としている．以下に例を示す.

図 2 のパターングラフ q とデータグラフ G において，射影演算をしない通常のパターン問合せと本研究で扱う射影演算を行うパターン問合せの違いについて述べる．通常のパターン問合せの場合，データグラフ G のパターングラフ q と同型の部分グラフが出力されるため，出力の結果は，

$\Sigma = \{a, c, d\}$
$V = \{v_1, v_4, v_5\}$
$E = \{(v_1,\ d,\ v_4), (v_1,\ c,\ v_5), (v_5,\ a,\ v_4)\}$

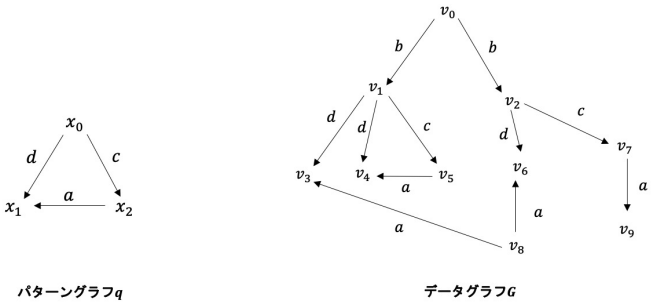

図 2　パターン問合せ

である部分グラフ全体が出力される．一方，図 2 のパターングラフ q で出力するノード変数として x_0 を指定した場合，データグラフ G のパターングラフ q と同型の部分グラフの中から x_0 に代入されるノード変数のみ出力されるため，出力の結果は v_1 となる．

2.3 Shape Expression Schema

本節では，Shape Expression Schema (以下 ShEx) を定義する．

ShEx は，RDF データ (グラフデータ) に対するスキーマ言語である．スキーマ言語とは，データの構造を定義する言語のことである．DTD や XML Schema などの多くのスキーマ言語において，型を定義する際には Regular Expression (正規表現) という規則を用いている．しかし，グラフデータでは一般にノード間の順序が考慮されないことが多いため，Regular Expression ではなく，Regular Bag Expression (以下 RBE) という規則を用いている．RBE と Regular Expression は，ほぼ同様の規則であるが，RBE は“ $\|$ ”を用いた連結の際に順序が無視されるという特性を持つ点が Regular Expression と異なる．RBE の規則は以下のようになっている．

- ϵ と $a \in \Sigma$ は RBE である．ここで，Σはラベルの集合を表す.
- $E_1, E_2, \cdots, E_k$ が RBE であるならば，$E_1|E_2|\cdots|E_k$ は RBE である．ここで,“ $|$ ”は選言を表す.
- $E_1, E_2, \cdots, E_k$ が RBE であるならば，$E_1 \| E_2 \| \cdots \| E_k$ は RBE である．ここで,“ $\|$ ”は順序を無視した連結を表す.
- E が RBE であるならば，$E^{[m,n]}$ は RBE である．ここで，$[m, n]$ は m 回以上 n 回以下の繰り返しを表す.

出現回数には以下の表記も用いる.

- ? ⋯ 0 回または 1 回
- $*$ ⋯ 0 回以上の繰り返し
- $+$ ⋯ 1 回以上の繰り返し

ShEx スキーマ S は，$S = (\Sigma, \Gamma, \delta)$ と表現される．ここで，Σ はラベルの集合，Γ は型の集合，δ は RBE を使用した型を定義する関数である．以下に例を 1 つ挙げる．

ShEx には，1 つのノードが 1 つの型のみをもつ single-type semantics と複数の型をもつことのできる multi-type semantics がある. 本研究では，single-type semantics についてのみ検討している．また, RBE による型の定義に選言“ $|$ ”を含まな

S ：

$\Sigma = \{a, b, c, d\}$
$\Gamma = \{t_0, t_1, t_2, t_3\}$
$\delta(t_0) = (a :: t_1)^* || b :: t_2 || c :: t_2$
$\delta(t_1) = a :: t_3 || c :: t_2$
$\delta(t_2) = d :: t_3$
$\delta(t_3) = \varepsilon$

い ShEx のみを対象としている．

2.4 ShEx の下での包含性判定問題

本節では，ShEx の下での包含性判定問題を定義する．

本研究では，問合せとしてパターングラフを用いている．q_1, q_2 をパターン問合せ，S を ShEx スキーマとする．S に妥当などのグラフデータに対しても，q_1 の解が q_2 の解に含まれるならば，q_2 は q_1 を包含するといい，「$q_1 \subseteq q_2$」と記述する．パターングラフ q_1, q_2 において，一方が他方を包含するかどうかを判定することを包含性判定と呼ぶ．

3 提 案 手 法

本章では，ShEx におけるパターン問合せの包含性判定アルゴリズムについて述べる．

3.1 前　　提

本節では，提案するアルゴリズムを実行する際に，前提としている事柄を定義する．

使用する ShEx スキーマとパターン問合せに用いる 2 つのパターングラフについて，以下のことを仮定する．

- 使用するパターングラフは，ShEx スキーマに対して充足可能な問合せであり，それぞれのノード変数には，対応する ShEx スキーマの型が 1 つわかっているものとする．
- 2 つのパターングラフによる問合せにおいて，両者のノード変数間の対応は既知であるとする．

パターン問合せの充足可能性判定については，文献 [5] にてアルゴリズムが提案されており，その提案アルゴリズムによってそれぞれのノード変数に対応する ShEx スキーマの型が判定できる．ノード変数に対応する ShEx スキーマの型が未知である場合，2 つのパターン間の共通部分グラフを求め，ノード変数間の対応を求める必要がある（共通部分グラフを求める問題は一般には NP 困難であるが，文献 [6] などのアルゴリズムが提案されている）．しかし，ノード変数に対応する ShEx スキーマの型が判明している場合，ShEx スキーマの型と射影演算する変数をもとに，両者のノード変数間の対応付けを容易に求めることができる．以上より，本研究における提案アルゴリズムでは上記の仮定を用いる．

この仮定により，ShEx スキーマにおけるパターングラフのエッジ e に対応する型は，入力ノード $N_s(e)$ と出力ノード $N_e(e)$ に対応する ShEx スキーマの型から判定することができる．例えば，ShEx スキーマの定義に「$\delta(t_0) = (a :: t_1)^* || b :: t_2$」という記述があり，エッジ e の入力ノード $N_s(e)$ の型が t_0，出力ノード $N_e(e)$ の型が t_1 である場合を考える．$N_s(e)$ と $N_e(e)$ に対応する ShEx スキーマの型から，エッジ e に対応する ShEx スキーマの型を「$\delta(t_0) = (a :: t_1)^*$」と表すことにする．また，エッジ e に対応する ShEx スキーマの型は，$*$ に修飾されていると表現する．

3.2 アルゴリズム

本節では，ShEx の下でのパターン問合せの包含性判定アルゴリズムに関して述べる．

前述のように，スキーマが定義されていないグラフデータに対してパターン問合せを行う場合や射影演算をしないパターン問合せであれば，部分グラフ同型問題で包含性の有無を判定できる．しかし，ShEx スキーマが定義されたグラフデータに対して，射影演算したパターン問合せの包含性を判定する場合においては，部分グラフ同型問題を解くだけでは正しく包含性を判定することができない．例として，以下の ShEx スキーマ $S_1 = (\Sigma, \Gamma, \delta)$ が定義されているグラフデータに対して，図 3 のパターングラフ q_1, q_2 を用いてパターン問合せを行う場合に，q_1 は q_2 包含されるかを考える．

S_1 ：

$\Sigma = \{a, b, c, d, e\}$
$\Gamma = \{t_0, t_1, t_2, t_3\}$
$\delta(t_0) = (a :: t_1)^+ || (b :: t_2)^? || c :: t_3$
$\delta(t_1) = (d :: t_1)^*$
$\delta(t_2) = (e :: t_1)^* \ || c :: t_3$
$\delta(t_3) = \varepsilon$

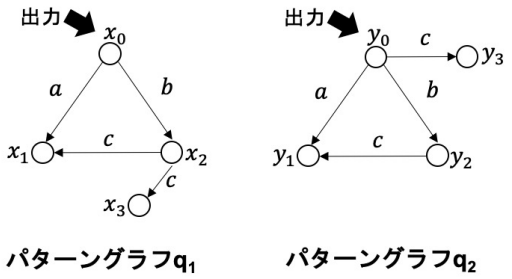

図 3　パターン問合せ q_1, q_2

ここで，q_1, q_2の ノードに対応する ShEx スキーマ S_1 の型は以下のようになる．以下では，ノード n に割り当てられる型を $\lambda(n)$ と表す．q_1, q_2 は互いに部分グラフではないが，ShEx スキーマの定義に基づき「$q_1 \subseteq q_2$」という包含性が認められる．

$\lambda(x_0) = t_0$	$\delta(y_0) = t_0$
$\lambda(x_1) = t_1$	$\delta(y_1) = t_1$
$\lambda(x_2) = t_2$	$\delta(y_2) = t_2$
$\lambda(x_3) = t_3$	$\delta(y_3) = t_3$
パターングラフ q_1	パターングラフ q_2

このように，部分グラフ同型問題では判定することのできない包含性を判定するためのアルゴリズムを提案する．この提案

アルゴリズムでは，パターングラフのノード数 3 以上の 2 連結成分 (biconnected component) について包含性を確認した後に，ShEx スキーマの定義に従い，包含される可能性のある一方のパターングラフに対してエッジを追加することで，部分グラフ同型問題のみでは判定できない包含性を判定する．

ここで，提案アルゴリズムの初めに，ノード数 3 以上の 2 連結成分の処理を行う理由について説明する．2 連結成分は，無向グラフにおいてノードを 1 つ削除しても接続された状態を保つことのできる極大部分グラフのことをいう．このように，2 連結成分は無向グラフに用いられる概念であるが，本研究では有向グラフにおけるエッジの向きを無視した際のノード間の繋がりに対してこの概念を用いている．パターン問合せにおいて，2 つのパターングラフが同じ変数とラベルを使っていても，ノード数が 3 以上の 2 連結成分のようにノード同士が繋がっているものとそうでないものでは解が異なる．例えば，図 4 のパターングラフ q, q' では，同じノード変数とラベルを使っているが，このパターングラフで問合せを行った際には，必ず q の解は q' の解に含まれる．これは，q にのみ x_2 から x_1 へのエッジが存在するため，データグラフに問合せを行う際に，部分グラフとなる条件が q' よりも q の方が多くなることに起因する．このように，ノード数 3 以上の 2 連結成分が含まれるパターングラフは，同じノード変数を用いてノード数 3 以上の 2 連結成分を持たないグラフよりも解が小さくなる．そのため，提案アルゴリズムでは初めにノード数 3 以上の 2 連結成分とその対応部分において包含性を確認する．

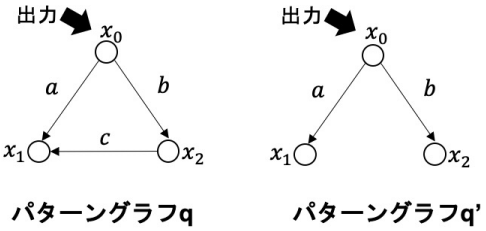

図 4　パターン問合せ q, q'

また，エッジの追加の際には，そのパターン問合せの解が変わることのないように，以下の条件をつけている．

- 他方のパターングラフに対応するエッジが存在する．
- エッジを追加しても、そのパターングラフにノード数が 3 以上の 2 連結成分が生じない．
- 対応する ShEx スキーマの型が?や*で修飾されない．
- 対応する ShEx スキーマの型が+で修飾されている場合，+の 2 個目以上のエッジではない．
- 対応する ShEx スキーマの型が {m},{m,n} で修飾されている場合，このエッジを追加しても m 個以下になる．
- 「逆向き」のエッジではない．

これらの条件を全て満たす場合にのみパターングラフにエッジを追加している．具体的な提案アルゴリズムを Algorithm 3.1 に示す．

Algorithm 3.1 CONTAINMENT DECISION

Input: ShEx schema $S = (\Sigma, \Gamma, \delta)$, query graph q_1, q_2
Output: 2 つの問合せにおける包含性の有無

```
A_{q1} :=CreateAdjacencyList(q1);
A_{q2} :=CreateAdjacencyList(q2);
P_{q1} :=GetArticulationPoints(A_{q1});
P_{q2} :=GetArticulationPoints(A_{q2});
C_{q1} :=SearchBiconnectedComponents(P_{q1}, q1);
C_{q2} :=SearchBiconnectedComponents(P_{q2}, q2);
if ∀x ∈ C_{q1} (|x| < 3) and ∀y ∈ C_{q2} (|y| < 3) then
    Result1 := AddEdge(q1, q2, S);
    Result2 := AddEdge(q2, q1, S);
else
    M_{q1} ← {c ∈ C_{q1} ||c| ≧ 3};
    M_{q2} ← {c ∈ C_{q2} ||c| ≧ 3};
    Result1 := IsInclude(M_{q1}, M_{q2}, q1, q2, S);
    Result2 := IsInclude(M_{q2}, M_{q1}, q2, q1, S);
end if
if Result1 = true then
    s1 ← " q1 ⊆ q2 "
else
    s1 ← " q1 ⊈ q2 "
end if
if Result2 = true then
    s2 ← " q2 ⊆ q1 "
else
    s2 ← " q2 ⊈ q1 "
end if
report (s1, s2)
```

Algorithm 3.2 AddEdge

Input: q_1, q_2, S

```
for each e ∈ E(q2) do
  if e ∉ E(q1) then
    if S における e に対応する型が * や ? で修飾されない then
      if N_s(e) ∉ V(q1) and N_e(e) ∈ V(q1) then
        break
      end if
      if S における e に対応する型が+で修飾 then
        if S における対応する型と入力ノードが e と同じエッジが q1 に存在する then
          break
        end if
      end if
      if S における e に対応する型が {m},{m,n} で修飾 then
        if S における対応する型と入力ノードが e と同じエッジが n 個 q1 に存在する then
          break
        end if
      end if
      if not SearchRoute(N_s(e), N_e(e), q1) then
        E(q1) ← E(q1) ∪ {e};
      end if
    end if
  end if
end for
if E(q2) ⊆ E(q1) then
  return true
else
  return false
end if
```

Algorithm 3.3 IsInclude

Input: $M_{q_1}, M_{q_2}, q_1, q_2, S$

```
1: for each m_2 ∈ M_{q_2} do
2:     if not ∃m_1 ∈ M_{q_1}(m_2 ⊆ m_1) then
3:         return false
4:     end if
5: end for
6: Result := AddEdge(q_1, q_2, S)
7: return Result
```

まず，1〜2 行目でパターングラフ q_1, q_2 の隣接リスト A_{q_1}, A_{q_2} をそれぞれ作成する．ここで呼び出している「CreateAdjacencyList」という関数は与えられたグラフの隣接リストを作成するための関数である．パターングラフ q_1, q_2 は有向グラフであるが，この隣接リストは 2 連結成分の探索のために用いるため，エッジの向きを考慮せずパターングラフ q_1, q_2 を無向グラフとみなして隣接リストを作成する．

次に 3〜4 行目で隣接リスト A_{q_1}, A_{q_2} を用いて関節点の探索を行う．ここで呼び出している「GetArticulationPoints」という関数は，隣接リストから関節点の集合を得るための関数である．ここで，関節点とは，連結のグラフからそのノードをグラフから取り除くとグラフが非連結になるノードである．この関数では，DFS 木と lowlink を使い関節点を探索している．この関数を用いて，パターングラフ q_1, q_2 それぞれの関節点の集合 P_{q_1}, P_{q_2} を得る．

5〜6 行目では，関節点の集合 P_{q_1},P_{q_2} を用いて，2 連結成分の探索を行う．ここで呼び出している関数「SearchBiconnectedComponent」は，与えられたグラフと関節点の集合を利用して 2 連結成分の集合を得る関数である．この関数では，関節点および接続するエッジをグラフから削除し，連結成分に分解し関節点に接続していたエッジを各連結成分に戻すことで 2 連結成分を求めている．この関数を用いてパターングラフ q_1, q_2 それぞれの 2 連結成分の集合 C_{q_1}, C_{q_2} を得る．

7〜15 行目では，2 連結成分の集合 C_{q_1}, C_{q_2} を用いて，パターングラフ q_1 と q_2 のどちらかにノード数が 3 以上の 2 連結成分があるか，もしくはどちらのパターングラフにもノード数が 3 以上の 2 連結成分が存在しないかを判定し，その後の処理を分ける．どちらのパターングラフにもノード数が 3 以上の 2 連結成分が存在しない場合は，8〜9 行目の処理を行う．8 行目は，関数「AddEdge」を利用し,「$q_1 \subseteq q_2$」という包含性の有無を判定し，$Result1$ に結果を格納する．9 行目も同様に,「$q_2 \subseteq q_1$」という包含性の有無を判定し，$Result2$ に結果を格納する．パターングラフ q_1 と q_2 のどちらかにノード数が 3 以上の 2 連結成分がある場合，11〜14 行目の処理を行う．11〜12 行目では，2 連結成分の集合 C_{q_1}, C_{q_2} のうちノード数が 3 以上の 2 連結成分の集合を，それぞれ M_{q_1}, M_{q_2} に格納する．13 行目は，関数「IsInclude」を利用し，与えられた M_{q_1}, M_{q_2} をもとに「$q_1 \subseteq q_2$」という包含性の有無を判定し，$Result1$ に結果を格納する．14 行目も同様に,「$q_2 \subseteq q_1$」という包含性の有無を判定し，$Result2$ に結果を格納する．

16〜20 行目は,「$q_1 \subseteq q_2$」という包含性の判定の結果が格納されている変数 $Result1$ の結果によって,「$q_1 \subseteq q_2$」であるか「$q_1 \not\subseteq q_2$」であるかを変数 s_1 に代入している．同様に 21〜25 行目は，変数 $Result2$ の結果によって,「$q_2 \subseteq q_1$」であるか「$q_2 \not\subseteq q_1$」を変数 s_2 に代入している．最後に，変数 s_1 と s_2 を出力して終了する．

Algorithm 3.2 に示した関数「AddEdge」では，引数として与えられたパターングラフ q_1, q_2 と ShEx スキーマ S を用いて，パターングラフ q_1 にエッジを追加し，その後に包含性判定を行う．具体的な処理は，以下のように行う．

- 1〜22 行目が，エッジの追加処理である．パターングラフ q_2 に含まれるエッジをそれぞれ q_1 に追加できるかを判定していく．以下，q_1, q_2 のエッジ集合を $E(q_1), E(q_2)$, q_1, q_2 のノード集合を $V(q_1), V(q_2)$ と表す．
- 1 行目では，$E(q_2)$ から q_2 に含まれるエッジをそれぞれエッジ e として扱う．2 行目では，$E(q_1)$ に含まれないエッジ e のみが次の処理に進むように定義している．3 行目では，ShEx スキーマ S においてエッジ e に対応する型が*や?で修飾されていない場合に次の処理へと進むように定義している．
- 以下，エッジ e の入力ノードを $N_s(e)$，出力ノードを $N_e(e)$ と表す．4〜6 行目で，$V(q_1)$ のノードとして $N_s(e)$ が存在せず，$V(q_1)$ のノードに $N_s(e)$ が存在する場合は，処理を中断させる．これは，追加するエッジ e の入力ノードが q_1 に存在せず，出力ノードが存在する場合は，必ず「逆向き」のエッジとなり問合せの解を小さくしてしまうために行う処理である．
- 7〜11 行目では，ShEx スキーマ S のエッジ e に対応する型が+で修飾されている場合において，対応する型と入力ノードがエッジ e と同じであるエッジが q_1 に存在した際に処理を中断させる．この処理は，ShEx スキーマ S のエッジ e に対応する型が+で修飾されている場合に，エッジを追加することで q_1 の解を変化させることがないように行う処理である．
- 12〜16 行目では，ShEx スキーマ S のエッジ e に対応する型が {m},{m,n} で修飾されている場合において，対応する型と入力ノードがエッジ e と同じであるエッジが q_1 に m 個存在した際に処理を中断させる．この処理は，ShEx スキーマ S のエッジ e に対応する型が {m},{m,n} で修飾されている場合に，エッジを追加することで q_1 の解を変化させることがないように行う処理である．
- 17〜19 行目では，エッジ e の入力ノード $N_s(e)$ から出力ノード $N_e(e)$ を繋いでいるエッジが既に q_1 に存在していないかを確認している．ここで呼び出している関数「SearchRoute」は，幅優先探索で $N_s(e)$ から $N_e(e)$ までの経路を探索し，経路があった場合に「true」を返す．この処理は，$N_s(e)$ から $N_e(e)$ を繋いでいるエッジが既に存在する場合，エッジ e を追加するとノード数が 3 以上の 2 連結成分が生じてしまうため，行う必要がある．18 行目で，ここまで処理を中断されることなく進んだエッジ e を q_1 のエッジ集合 $E(q_1)$ に加える．
- 23〜27 行目は，ShEx スキーマの定義に従いエッジを追加した q_1 のエッジ集合 $E(q_1)$ に q_2 のエッジ集合 $E(q_2)$ が包含されるかを判定する．包含された場合は「true」を返し，包含

されない場合は「false」を返す.

Algorithm 3.3 に示した関数「IsInslude」では，q_1, q_2 のノード数 3 以上の 2 連結成分の集合 M_{q_1}, M_{q_2} が与えられた際に M_{q_1}, M_{q_2} のそれぞれの要素に包含性があるかを判定し，その後に関数「AddEdge」を用いてグラフ全体の包含性判定を行う．具体的な処理は，以下のように行う.

- 1〜5 行目で，Mq_2 に含まれる全ての 2 連結成分が M_{q_1} に含まれるいずれかの 2 連結成分の部分グラフになるかどうかを判定している．以下，M_{q_1}, M_{q_2} の要素であるそれぞれの 2 連結成分を m_1, m_2 とする.
- 2〜4 行目では，m_1 の部分グラフとならない m_2 が存在した場合に「false」を返すようにしている．ここで，全ての m_2 においていずれかの m_1 の部分グラフにならない場合，q_1 の解で q_2 の解に含まれないものが存在してしまうため，必ず「$q_1 \not\subseteq q_2$」になる． したがってこのような場合は，包含性判定の結果として「false」を返す.
- 6 行目では，1〜5 行目の処理を通過している問合せに対して，M_{q_1}, M_{q_2} 以外の部分での包含性を判定するために関数「AddEdge」を用いている．その結果を変数 *Result* に格納し，7 行目で *Result* を返している.

ここからは具体的な例を用いて，関数「IsInclude」および関数「AddEdge」の処理の説明を行う．前述の ShEx スキーマ S_1 の下における図 3 のパターン問合せ q_1, q_2 について「$q_1 \subseteq q_2$」であるかを考える．ここで関数「IsInclude」の引数である M_{q_1}, M_{q_2} は，それぞれノード変数が「x_0, x_1, x_2」,「y_0, y_1, y_2」である集合のみで構成される．このノード数 3 以上の 2 連結成分を m_1, m_2 として，図 5 に示す．この m_2 は m_1 の部分グラフとなるため，次の処理に進む.

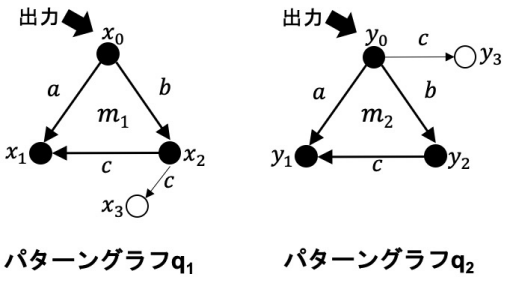

図 5　q_1, q_2 の 2 連結成分

次の処理では，関数「AddEdge」を用いて「$q_1 \subseteq q_2$」という包含性を判定する．関数「AddEdge」では，q_2 のエッジで q_1 に対応するものがないエッジ e を q_1 に追加できるかを確認していく．このエッジ e にあたるエッジは (y_0, c, y_3) のみであるため，以下からこのエッジをエッジ e と表す．ここで，エッジ e に対応する ShEx スキーマ S_1 の型は,「$\delta(t_0) = c :: t_3$」である．この ShEx スキーマ S_1 における エッジ e に対応する型は，*や＋などで修飾されていないため，処理を中断することなく進む．次に，エッジ e の入力ノードに対応する q_1 のノードから出力ノードに対応する q_1 のノードまでの経路を探索する．エッジ e の入力ノードである y_0 に対応する q_1 のノードは x_0 であり，出力ノードの y_3 に対応するノードは q_1 に存在しない．そのため，経路は存在しないので，エッジ e を q_1 に追加することができる．q_2 とエッジ e に追加した q_1 を図 6 に示す.

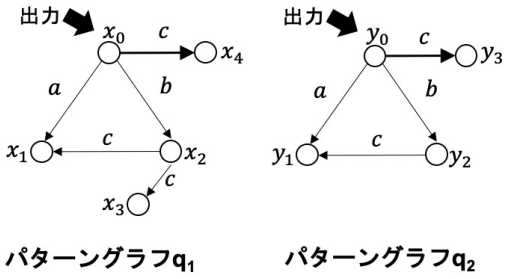

図 6　q_1, q_2 の包含性判定

最後に q1 にエッジを追加した図 6 の状態で，q_1 のエッジ集合 $E(q_1)$ に q_2 のエッジ集合 $E(q_2)$ が包含されるかを判定する．この時，$E(q_2)$ の全てのエッジに対応するエッジが $E(q_1)$ に存在するため,「$q_1 \subseteq q_2$」と判定できる.

以下，アルゴリズムの正当性を示す.

定理 1. Algorithm 3.1 において「$q_1 \subseteq q_2$」と判定するときかつそのときのみ $q_1 \subseteq q_2$ である.

証明の概要.
(⇒)　Algorithm3.1 で「$q_1 \subseteq q_2$」と判定したと仮定する．Algorithm 3.2 によって q_1 にエッジを追加したものを $q_1{}'$ とする．q_2 が $q_1{}'$ の部分グラフになっているため $q_1{}' \subseteq q_2$ である．Algorithm 3.2 においてエッジを追加する際の条件により，q_1 と $q_1{}'$ の解は等しい．よって，$q_1 \subseteq q_2$ である.
(⇐)　$q_1 \subseteq q_2$ ならば，アルゴリズムは「$q_1 \subseteq q_2$」と判定することを証明するために，この命題の対偶が真であることを示す．Algorithm3.1 で「$q_1 \not\subseteq q_2$」と判定したと仮定する．このとき，q_2 には含まれるが $q_1{}'$ には含まれないエッジが必ず存在する．このエッジは Algorithm 3.2 のエッジの追加のための条件を満たさない．そのため，q_2 にはマッチしないが $q_1{}'$ にはマッチするパターンが必ず存在し得る．よって， $q_1{}' = q_1 \not\subseteq q_2$ である.
□

4　評 価 実 験

本章では，提案アルゴリズムに関する評価実験について述べる.

4.1　概　　要

はじめに，前章で述べた提案アルゴリズムを Python で実装した．次に，グラフデータの構造を定義した ShEx スキーマを用意した．その後，ShEx スキーマに対して充足可能なパターングラフを作成し，その ShEx スキーマとパターングラフを用いて提案アルゴリズムの実行時間を計測した.

使用した ShEx スキーマは，Wikidata の Entity Schema と SP^{2}Bench で生成した RDF データに対して定義したスキー

マである．Wikidata [7] は，Wikimedia 財団が運営するプロジェクトの 1 つとして提供されている，web 上で誰もが編集可能な構造化データのデータベースである．Entity Schema は，Wikidata の各項目やプロパティの関係性を定義するスキーマとなっている．評価実験では，「edition of a written work(E36)」という名称の Entity Schema を用いた．これは，Wikidata 上の著作物の版や作品・著作者に関するデータに対して定義されたスキーマである．このスキーマを元に，提案アルゴリズムで扱うために ShEx スキーマを定義した．

SP^{2}Bench [8] とは，コンピュータ科学に関する書誌学 web サイトのデータベースである DBLP に基づき，指定したトリプル数もしくはデータサイズの RDF データを作成する SPARQL の包括的なベンチマークソフトである．トリプルとは，RDF データのリソースに関する関係情報を表現する「主語，述語，目的語」の 3 つの要素のことである．また，トリプルはグラフのエッジに相当するため，本研究においてはトリプルをエッジとみなしている．図 7 に SP^{2}Bench のデータ構造，図 8 に DBLP と SP^{2}Bench のラベルの対応関係を表した図を示す (いずれも文献 [8] から引用している)．SP^{2}Bench で作成されるデータには，スキーマが定義されていない．そのため，SP^{2}Bench で作成したデータに対して，前述のグラフ構造とラベルに基づき，提案アルゴリズムで扱うために ShEx スキーマを定義した．

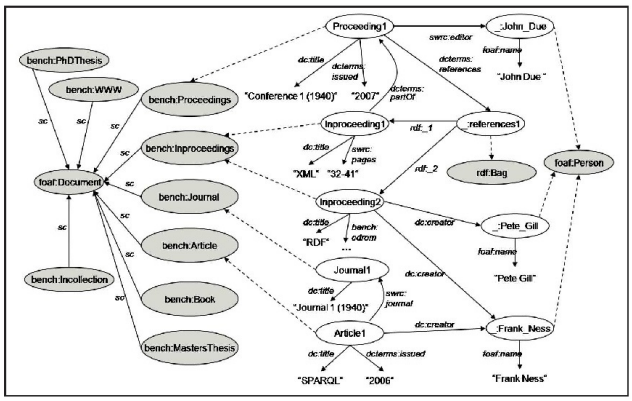

図 7　SP^{2}Bench のデータ構造

attribute	mapped to prop.	refers to
address	swrc:address	xsd:string
author	dc:creator	foaf:Person
booktitle	bench:booktitle	xsd:string
cdrom	bench:cdrom	xsd:string
chapter	swrc:chapter	xsd:integer
cite	dcterms:references	foaf:Document
crossref	dcterms:partOf	foaf:Document
editor	swrc:editor	foaf:Person
ee	rdfs:seeAlso	xsd:string
isbn	swrc:isbn	xsd:string
journal	swrc:journal	bench:Journal
month	swrc:month	xsd:integer
note	bench:note	xsd:string
number	swrc:number	xsd:integer
page	swrc:pages	xsd:string
publisher	dc:publisher	xsd:string
school	dc:publisher	xsd:string
series	swrc:series	xsd:integer
title	dc:title	xsd:string
url	foaf:homepage	xsd:string
volume	swrc:volume	xsd:integer
year	dcterms:issued	xsd:integer

図 8　DBLP と SP^{2}Bench のラベルの対応関係

利用するパターングラフは，ShEx スキーマごとに 15 個ずつ作成した．1 つのパターングラフにおけるエッジ数は 3 から 7 であり，グラフサイズごとに 3 つずつのパターングラフとなるように作成した．また各グラフサイズには，ノード数が 3 以上の 2 連結成分を含んだパターングラフが 1 つずつ存在する．15 個のパターングラフから 2 つを取り出す組み合わせを全通り行うため，1 つの ShEx スキーマにつき 105 通りの組み合わせを用いて提案アルゴリズムを実行した．パターングラフ作成の際に射影演算するノードの型を指定しており，提案アルゴリズムの実行の際には，その型を持つノードを起点として 2 つのパターングラフのノードに対応関係を持たせている．

評価実験の環境は以下の通りである．

プロセッサ　1.4 GHz クアッドコア Intel Core i5
主記憶　8GB
OS　macOS Catalina 10.15.5
使用言語　Python 3.7.7

4.2 結　　果

前節で述べた ShEx スキーマ，パターングラフを用いて，提案アルゴリズムにより包含性判定を行い，実行時間を計測した．実行時間の計測は Python の Time モジュールを使用して計測しており，計測結果の単位は秒である．ShEx スキーマごとに 105 通りの組み合わせの実行時間の平均を計算した．結果を表 1 に示す．また，パターングラフにノード数 3 以上の 2 連結成分を含むか否かという特徴ごとに，場合わけして計算した実行時間の平均を表 2 に示す．加えて，パターングラフのサイズごとに実行時間の平均を計算した．表 3 に，Wikidata の Entity Schema において，パターングラフのサイズごとに計算した実行時間の平均を示す．表 4 に，SP^{2}Bench で生成した RDF データに対して定義したスキーマにおいて，パターングラフのサイズごとに計算した実行時間の平均を示す．

表 1　平均実行時間

ShEx スキーマ	Wikidata	SP^{2}Bench
実行時間の平均	1.89×10^{-4}	2.15×10^{-4}

表 2　パターングラフに 2 連結成分を含むか否かで場合分けした場合の平均実行時間

2 連結成分	個数	Wikidata	SP^{2}Bench
両方あり	10	1.14×10^{-4}	1.34×10^{-4}
片方あり	50	1.67×10^{-4}	1.90×10^{-4}
両方なし	45	2.30×10^{-4}	2.62×10^{-4}

表 3　Wikidata の ShEx スキーマにおけるパターングラフのサイズごとの平均実行時間

7					2.47×10^{-4}
6				2.27×10^{-4}	2.40×10^{-4}
5			1.94×10^{-4}	2.11×10^{-4}	2.23×10^{-4}
4		1.55×10^{-4}	1.79×10^{-4}	1.84×10^{-4}	2.02×10^{-4}
3	1.30×10^{-4}	1.40×10^{-4}	1.58×10^{-4}	1.73×10^{-4}	1.78×10^{-4}
	3	4	5	6	7

表 4　SP^{2}Bench の ShEx スキーマにおけるパターングラフのサイズごとの平均実行時間

7					2.97×10^{-4}
6				2.86×10^{-4}	2.63×10^{-4}
5			2.19×10^{-4}	2.08×10^{-4}	2.75×10^{-4}
4		1.66×10^{-4}	1.85×10^{-4}	2.28×10^{-4}	2.23×10^{-4}
3	1.59×10^{-4}	1.82×10^{-4}	1.65×10^{-4}	1.69×10^{-4}	2.38×10^{-4}
	3	4	5	6	7

4.3 考　　察

前節で示した結果を見ると，提案アルゴリズムによって，ShEx スキーマの下でのパターン問合せの包含性判定がわずかな時間で行えることがわかる．また，パターングラフにノード数 3 以上の 2 連結成分を含む場合に，実行時間が短くなる傾向が見られた．提案アルゴリズムでは，パターングラフにノード数 3 以上の 2 連結成分を含む場合に処理を変えており，関数「IsInclude」の処理時に包含性がないことが確認された場合，エッジの追加処理を行わないため，実行時間が短くなると考えられる．加えて，パターングラフのサイズによっても，実行時間が変化する傾向が見られた．おおむね 2 つのパターングラフのサイズが大きくなるほど，実行時間が長くなっていた．これは提案アルゴリズムでは，隣接リストの作成やエッジの追加処理など，エッジの数が多いほど時間がかかる処理が存在しているためと考えられる．

5 む　す　び

本論文では，ShEx スキーマが定義されたグラフデータに対して行うパターン問合せの包含性判定を行うアルゴリズムを提案した．より具体的には，ShEx スキーマと射影演算した 2 つのパターン問合せが与えられた時，それらのパターン問合せの解に包含性があるかを判定する手法を提案した．その手法は，ShEx のスキーマ定義に従い，片方のパターン問合せに対してエッジを追加することで部分グラフ同型問題では判定できない包含性を判定する．また，評価実験として ShEx スキーマとその下で充足可能なパターン問合せに対して，Python 言語で実装した提案アルゴリズムを実行し，実行時間を計測した．その結果，わずかな時間で包含性の判定を正確に行うことできることを確認した．また，パターン問合せのサイズや 2 連結成分の有無という特徴によって，実行時間が変化することもまた確認した．

今後の課題としては，本論文では ShEx スキーマの型の定義に選言“ | ”を含まない場合に限定したアルゴリズムを提案したため，ShEx スキーマの型の定義に選言“ | ”を含む場合にも対応させることが挙げられる．また，本論文では single-type semantics についてのみ検討したため，multi-type semansics にも対応させることも課題である．今後は，これらの課題を解決したアルゴリズムに改善予定である．

文　　献

[1] Wood,P.T.“Containment for XPath fragments under DTD constraints. Lecture notes in computer science”. 2003, vol. 2572, pp. 300-314.

[2] Reinhard Pichler, Sebastian Skritek. “Containment and equivalence of well-designed SPARQL”. Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. 2014, pp. 39-50.

[3] World Wide Web Consortium(W3C). “Shape Expressions (ShEx) 2.1 Primer”. https://shex.io/shex-primer/ , (accessed 2020-12-01).

[4] World Wide Web Consortium(W3C). “RDF Schema 1.1”. https://www.w3.org/TR/rdf-schema/ , (accessed 2020-12-01).

[5] Shiori Matsuoka, Nobutaka Suzuki. “Detecting Unsatisfiable Pattern Queries under Shape Expression Schema”. Proceedings of the 16th International Conference on Web and Information Systems and Technologies (WEBIST 2020). 2020, pp. 285-291.

[6] Evgeny B. Krissinel, Kim Henrick,“Common Subgraph Isomorphism Detection by Backtracking Search,” Journal of Software: Practice and Experience, 34(6), pp.591-607.

[7] Wikimedia Foundation.“Wikidata”. https://www.wikidata.org/wiki/Wikidata:Main_Page, (accessed 2020-12-01).

[8] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. “SP^{2}Bench: a SPARQL Performance Benchmark,” Proc. ICDE, 2009, pp. 371-393.