DEIM Forum 2019 F5-2

Shape Expression Schemaにおけるパターン問合せの充足可能性判定

松岡　栞†　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: †s1511560@s.tsukuba.ac.jp, ††nsuzuki@slis.tsukuba.ac.jp

あらまし　Shape Expression Schema(ShEx) はグラフデータのスキーマを記述するために新たに提案されたスキーマ言語である．一般に，グラフデータは莫大なデータ量をもち，このようなデータに充足不能なパターン問合せを行う場合，その検索は明らかに無駄である．そのため，充足不能なパターンを ShEx の定義から発見できることが望ましい．しかし，ShEx には選言による分岐が存在し，部分グラフ同型問題を適用して解くのは不十分である．そこで本研究では部分グラフ同型問題の解法の一つである Ullman Algorithm を拡張し，適応させた．考案アルゴリズムを用いることで問合せ時間を短時間で済ますことが可能だと確認した．

キーワード　Shape Expression Schema，パターン問合せ，充足可能性判定

1 はじめに

近年，RDF データ/グラフデータは急速に増加し，様々なところで用いられている．グラフデータは規模が大きいことが多いため，より効率の良い問合せ処理が必要とされている．スキーマは RDB や XML などではよく活用され，技術も確立しているが，グラフデータのスキーマの活用についてはまだ途上の段階である．グラフデータのスキーマは多様であり，例として RDF Schema [1] やグラフスキーマが挙げられる．RDF Schema は従来の RDF のスキーマ言語として提案されているものである．しかし，オントロジ記述言語としての性格が強く，スキーマ言語としてのセマンティクスが W3C で定義されていない．さらに，グラフスキーマはスキーマをグラフとして表現するものであるが繰り返しや入れ子といった表現力が低い．そこで，グラフデータの普及に伴い，RDF Schema やグラフスキーマより適切に記述することの可能なスキーマが必要となり，Shape Expression Schema（以下 ShEx）[2] が誕生した．ShEx は型を Regular Bag Expression(以下 RBE) という規則に基づいて表し，グラフの各ノードに対し型を割り当てられるスキーマである．

パターン問合せは，問合せとしてグラフの断片（パターン）を指定してグラフからそのパターンにマッチする部分を検索するもので，グラフに対する最も基本的かつ重要な問合せである．一般にグラフデータは莫大なデータ量をもつ．このようなデータに対してパターン問合せを行う場合，その莫大なデータの中からマッチするパターンを検索しなければならない．パターン P と ShEx S に対して，S に妥当かつ P を含むようなグラフデータ（RDF データ）が存在しなかった場合，すなわち充足不能であった場合は膨大なデータを検索したのちに解が存在しないという結果が返される．しかし，その検索は明らかに無駄である．また，パターンがマッチするかどうか調べる場合，一般には部分グラフ同型問題を解けばよいとされるが，ShEx は選言「|」を用いて，「$E_1|E_2$」のように E_1 または E_2 に一致するという型を定義できる．検索する ShEx に選言「|」による分岐が存在した場合，部分グラフ同型問題では解くことができない．なぜなら，分岐しているうちの一方のエッジが選択された場合，他方のエッジは以降のマッチングにおいて利用することができないからである．

そこで本研究では，グラフデータと比較して容量の著しく軽い ShEx を修正 AND/OR グラフで表現し，パターン問合せの充足可能性を判定するアルゴリズムを提案する．より具体的には，まず ShEx を最もサイズの小さい修正 AND/OR グラフで表現し，その中で選言「|」による分岐をもつノードを抽出する．その後，部分グラフ同型問題の解法の一つである Ullman Algorithm [3] を元に，パターンと修正 AND/OR グラフとのマッチングを行う．この際，Ullman Algorithm に選言「|」に対応した判定を加え，実行する．マッチング中のノードが選言「|」による分岐をもつノードであった場合には，同時に分岐しているエッジを使用せずにマッチング可能かの判定，マッチング中のノードに接続している元のノードが選言「|」による分岐をもつノードであった場合には，ほかの分岐先のエッジを使用済みでないかの判定を行う．パターンのノード全てとマッチすることが可能であれば，充足可能として処理を終了するアルゴリズムである．本研究の目的が達成されれば，大規模なグラフデータをより効率よく問合せることが可能となり，グラフデータを利用する研究者・開発者にとって有用と考えられる．著者の知る限り，ShEx の下でのパターン問合せ充足可能性問題を解くためのアルゴリズムはこれまで提案されていない．

提案アルゴリズムを実装し，充足不能なパターンを対象として評価実験を行った．その結果，提案アルゴリズムが充足可能性判定を行えること，充足不能な場合に処理時間を少なく済ませられることを確認した．

本稿の構成は以下の通りである．第 2 章では，グラフ，ShEx，充足可能性問題の定義を行う．第 3 章では，ShEx に対する前処理および提案する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，本研究のまとめについて述べる．

2 諸 定 義

本章では本研究で扱うグラフの概要，Shape Expression Schema，充足可能性問題について述べる．

2.1 グラフの概要

本研究ではラベル付き有向グラフ (以下グラフ) を利用している．グラフを $G = (V, E)$ 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．以下，エッジ (n, a, n') を $n \xrightarrow{a} n'$ と表記することがある．例として，図 1 に簡単なグラフ G_0 を示す．

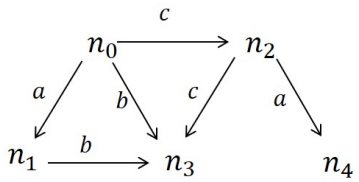

図 1 グラフ G_0

ここで，グラフ G_0 は，

$$\begin{aligned} \Sigma &= \{a, b, c\} \\ V &= \{n_0, n_1, n_2, n_3, n_4\} \\ E &= \{n_0 \xrightarrow{a} n_1, n_0 \xrightarrow{c} n_2, n_0 \xrightarrow{b} n_3, n_1 \xrightarrow{b} n_3, \\ &\quad n_2 \xrightarrow{c} n_3, n_2 \xrightarrow{a} n_4\} \end{aligned}$$

である．また，エッジを出力しているノードを出力ノード，エッジの向かう先である受け取り側のノードを入力ノードと呼称する．各ノードは出力ノードである場合，入力ノードである場合，さらに，出力ノードであり入力ノードである場合がある．

一般的な AND/OR グラフでは同時に存在する「AND」のエッジに円弧を描くことでグラフ中の分岐を可視化しているが，本研究では分岐を表す「OR」のエッジに注目するため，図 2 のように「OR」のエッジに円弧を描いている．これを修正 AND/OR グラフと呼ぶことにする．

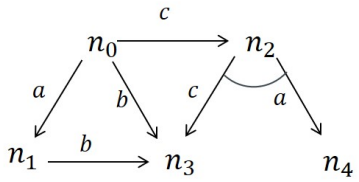

図 2 グラフ G_0

2.2 Shape Expression Schema

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

XML のスキーマである DTD や XML Schema といった多くのスキーマでは，型を定義する際に Regular Expression(正規表現) を用いる．ShEx では，Regular Expression ではなく Regular Bag Expression(以下 RBE) を用いている．Regular Expression と違い，RBE は連結“||”において順序が無視される．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．
- $E_1, E_2, \cdots, E_k$ 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，$E_1 \parallel E_2 \parallel \cdots \parallel E_k$ 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，$E^{[n,m]}$ は RBE である．ここで，$[n, m]$ は n 回以上 m 回以下の繰り返しを表す．

ShEx 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．例として 1 つ以下に挙げる．

$$\begin{aligned} S: \\ \Sigma &= \{a, b, c\} \\ \Gamma &= \{t_0, t_1, t_2, t_3, t_4\} \\ \delta(t_0) &= (a :: t_1 | b :: t_2 | c :: t_2)^* \\ \delta(t_1) &= b :: t_3 | c :: t_4 \\ \delta(t_2) &= c :: t_3 \\ \delta(t_3) &= \varepsilon \\ \delta(t_4) &= a :: t_3 \end{aligned}$$

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 とがある．本研究では single type semantics についてのみ検討している．

第 3 節で述べる提案アルゴリズムでは，ShEx をグラフとして表した上で充足可能性を判定する．ShEx $S = (\Sigma, \Gamma, \delta)$ であった場合，作成されるスキーマグラフ $G_S = (V_S, E_S)$ は以下のように定義される．

$$\begin{aligned} V_S &= \Gamma \\ E_S &= \{(t \xrightarrow{a} t') \mid \delta(t) \text{ に } a :: t' \text{ が出現する } \} \end{aligned}$$

具体的には，先に述べた ShEx S から作成されるスキーマグラフ $G_S = (V_S, E_S)$ は，

$$\begin{aligned} V_S &= \{t_0, t_1, t_2, t_3, t_4\} \\ E_S &= \{t_0 \xrightarrow{a} t_1, t_0 \xrightarrow{b} t_2, t_0 \xrightarrow{c} t_2, t_1 \xrightarrow{b} t_3, t_1 \xrightarrow{c} t_4, \\ &\quad t_2 \xrightarrow{c} t_3, t_4 \xrightarrow{a} t_3\} \end{aligned}$$

となる．

2.3 充足可能性問題

本節では充足可能性問題を定義する．

本研究では，問合せとしてパターンを考える．ここで，パターンもグラフである．パターン問合せとは，グラフ問合せにおいて最も基本的かつ重要なもので，データグラフにおいてパターンと同型である（マッチする）部分グラフが解となる．問合せ q，ShEx S に対して，「S に妥当かつ q の問合せ結果が空でな

いグラフデータが存在する」と q は S の下で充足可能であるという．このとき，問合せが充足可能であるかどうかを判定する問題を充足可能性問題という．2.2 節で述べたような ShEx において，$\delta(t_1)$ をもつノード n_1，$\delta(t_3)$ をもつノード n_2，$\delta(t_4)$ をもつノード n_3 が存在した際，$n_1 \xrightarrow{b} n_2$ かつ $n_1 \xrightarrow{c} n_3$ であるパターンはそれぞれが同時に存在することが選言「|」で接続されていることから不可能であり，充足不能となる．なお，部分グラフ同型問題は NP 困難であるため，同様に本研究の充足可能性問題も NP 困難である．

3 提 案 手 法

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

3.1 前 処 理

本節では，充足可能性判定の際に用いる ShEx を元に作成するグラフについて述べる．

本研究においては充足可能性の判定を行うことが目的である．したがって，充足可能性の判定においては，エッジの出現回数は考慮する必要がないため，型の定義について，0 回以上の繰り返しを表現する「$*$」，0 または 1 回の出現を表現する「?」，1 回以上の繰り返しを表現する「+」についてはいずれも 1 回のみの繰り返しとする．さらに，$[n, m]$ については n 回の出現として作成する．選言「|」が存在する場合は区別のために円弧を描くが，選言「|」で接続されたノードが繰り返し処理の中に含まれていた場合は充足可能性には影響を及ぼさないため，円弧を描かず各エッジが一本ずつ出力することとなる．例として以下を挙げる．

型 $\delta(t_0) = (a :: t_1 | b :: t_2 | c :: t_2)^*$ から
$t_0 \xrightarrow{a} t_1, t_0 \xrightarrow{b} t_2, t_0 \xrightarrow{c} t_2$
の三本のエッジが作成される

3.2 アルゴリズム

本節では，ShEx におけるパターン問合せの充足可能性判定アルゴリズムについて述べる．

前述のように，ShEx は修正 AND/OR グラフとして表すことができる．したがって，もし ShEx が選言「|」を含まないのであれば， パターン問合せが ShEx のグラフに部分グラフとして出現するかどうかを判定する，すなわち，部分グラフ同型問題を解くことにより，充足可能性を判定できる．しかし，ShEx が選言「|」を含む場合，部分グラフ同型問題をそのまま解くだけでは正しく判定を行うことができない．なぜならば，選言「|」を含むエッジを同時に選択する必要のあるパターンで問合せを行った場合，どちらか一方のエッジを採択した時点でもう一方のエッジを利用することは選言「|」によって阻害されてしまうが，使用可能であるかを判定することができず，存在してはいけないエッジを使用してしまうことがあるからだ．そこで，部分グラフ同型問題を解くアルゴリズムを拡張することで，ShEx におけるパターン問合せの充足可能性判定ができるようにする．

本研究で提案するアルゴリズムは，部分グラフ同型問題の解法の一つである Ullman Algorithm に判定を追加したものである．Algorithm 3.2.1 に文献 [4] の表記に従って Ullman Algorithm を引用する．

Algorithm 3.1 GENERIC QUERY PROC

Input: query graph q, data graph g
Output: all subgraph isomorphisms of q in g

```
1:  M := ∅;
2:  for each u ∈ V(q) do
3:      C(u) := FilterCandidates (q, g, u, ...);
        [[∀v ∈ C(u)((v ∈ V(g)) ∧ (L(u) ⊆ L(v)))]]
4:      if C(u) = ∅ then
5:          return;
6:      end if
7:  end for
8:  SubgraphSearch (q, g, M, ...)
Subroutine SubgraphSearch (q, g, M, ...)
1:  if |M| = |V(q)| then
2:      report M;
3:  else
4:      u := NextqueryVertex (...);
        [[u ∈ V(q) ∧ ∀(u', v) ∈ M(u' ≠ u)]]
5:      C_R := RefineCandidates (M, u, C(u), ...);
        [[C_R ⊆ C(u)]]
6:      for each v ∈ C_R such that v is not yet matched do
7:          if IsJoinable (q, g, M, u, v, ...) then
8:              [[∀(u', v') ∈ M((u, u') ∈ E(q) ==> (V, V') ∈ E(g) ∧
                L(u, u') = L(v, v'))]]
9:              UpdateState (M, u, v, ...);
                [[(u, v) ∈ M]]
10:             SubgraphSearch (q, g, M, ...);
11:             RestoreState (M, u, v, ...);
                [[(u, v) ∉ M]]
12:         end if
13:     end for
14: end if
```

続いて，Algorithm 3.2.2 に提案アルゴリズムを示す．

提案するアルゴリズムは Algorithm3.2.1 中に選言「|」にかかわる判定を 3 つ追加する．1 つ目は,「FilterCandidates」後に選言「|」を含むノードを抽出する関数「BranchExtraction」である．2 つ目は,「IsJoinable」の判定内容にマッチング中のノードが選言「|」を含むノードであるか判定する関数「IsBranch-Former」を，3 つ目は，同様の位置にマッチング中のノードと隣接するマッチング中のノードが選言「|」を含むノードであるか判定する関数「IsBranchDestination」である．

Algorithm 3.2 SATISFIABILITY DECISION

Input: query graph q, schema graph g
Output: 充足可能かどうか

```
1: M := ∅;
2: for each u ∈ V(q) do
3:    C(u) := FilterCandidates (q, g, u, ...);
      [[∀v ∈ C(u)((v ∈ V(g)) ∧ (L(u) ⊆ L(v)))]]
4:    B := BranchExtraction (u, ...);
5:    if C(u) = ∅ then
6:       return;
7:    end if
8: end for
9: SubgraphSearch (q, g, M, ...)
```

Subroutine SubgraphSearch $(q, g, M, ...)$

```
1: if |M| = |V(q)| then
2:    report 充足可能;
3: else
4:    u := NextqueryVertex (...);
      [[u ∈ V(q) ∧ ∀(u', v) ∈ M(u' ≠ u)]]
5:    C_R := RefineCandidates (M, u, C(u), ...);
      [[C_R ⊆ C(u)]]
6:    for each v ∈ C_R such that v is not yet matched do
7:       if IsJoinable (q, g, M, u, v, ...) then
8:          [[∀(u', v') ∈ M((u, u') ∈ E(q) ==> (V, V') ∈ E(g) ∧
            L(u, u') = L(v, v'))]]
9:          if IsBranchFormer (q, g, M, u, v, ...) then
10:            if IsBranchDestination (q, g, M, u, v) then
11:               UpdateState (M, u, v, ...);
                  [[(u, v) ∈ M]]
12:               SubgraphSearch (q, g, M, ...);
13:               RestoreState (M, u, v, ...);
                  [[(u, v) ∉ M]]
14:            end if
15:         end if
16:      end if
17:   end for
18:   report 充足不能;
19: end if
```

2〜8 行目で，パターン問合せ q の各ノード u に対して，u にマッチし得る候補ノード集合 $C(u)$ を求める．候補ノード v から選言「|」でエッジが出力されている場合，v を B に追加する．9 行目で SubgraphSearch により q を再帰的に探索し，パターン問合せを実行する．SubgraphSearch では 1 行目でパターングラフのノード数とマッチしたノード集合 M のサイズが一致しているか判定する．一致していれば「充足可能」と出力し終了，不一致であれば 3 行目以降へと進む．4〜6 行目で以降の判定を行うスキーマグラフ中の候補ノード v を決定する．7〜10 行目で候補ノード v が正しいか判定を行う．マッチした場合は 11 行目で集合 M に組み合わせを追加，12 行目で再び SubgraphSearch を呼び出す．マッチしなかった場合は 13 行目で探索していた組み合わせを削除し，次の候補ノードを探索する．すべての候補ノードを探索し，かつ q のノード数と集合 M のサイズが不一致であれば「充足不能」と出力し，終了する．

各関数について説明していく．

- FilterCandidates は入出力エッジがマッチング中のパターンと一致する可能性のある，前処理で作成されたグラフ（以下スキーマグラフ）のノード集合をもとめる関数である．
- BranchExtraction はスキーマグラフ中の選言「|」で分岐しているエッジの接続元であるノード集合をもとめる関数である．
- NextQueryVertex はパターングラフ中のノード一つを選択し，マッチングを進めるために取得する関数である．前述した FilterCandidates を通して作成されたパターングラフの各ノードごとの集合 $C(u)$ から順に選択される．
- RefineCandidates はスキーマグラフ中の候補ノードから NextQueryVertex で取得した u よりも次数の小さい頂点を候補から外す関数である．アルゴリズムの高速化に大きく関わる関数であるが，本研究が対象とする ShEx のスキーマグラフはサイズが極めて小さいため現時点ではあまり重要視していない．
- IsJoinable は隣接する頂点について以降の判定を行っていく関数である．u と隣接する頂点がスキーマグラフ中のいずれのノードともマッチングしていない場合は判定をスキップして次の処理へと進む．マッチングしていた場合はマッチング済のノード (u', v') とマッチング中のノード (u, v) とを接続するエッジが一致しているかを確認していく．
- IsBranchFormer はマッチング中のノード v について，BranchExtraction で作成したノード集合 B 中に含まれているか判定する関数である．含まれていない場合はいずれのエッジを選択しても充足可能性は損なわれないため，次の処理へと進む．しかし，含まれていた場合，すでにマッチング済のノードへと接続するエッジが選言「|」で分岐しているうちの二本以上であった場合，それらのエッジを同時に使用してはいけないためにマッチング中のノードは不適切であるとなるため，次のノード候補へと戻り，処理を再開させる．
- IsBranchDestination はマッチング済のノードであり，マッチング中のノードと隣接している頂点 v' について，BranchExtraction で作成したノード集合 B 中に含まれているか判定する関数である．含まれていない場合は，マッチング中のノードが加わっても充足可能性は損なわれないため，次の処理へと進む．しかし，含まれていた場合はほかの選言「|」で分岐しているエッジを使用しているか否かの判定を行う必要がある．マッチング済でマッチング中のノードと隣接しているノードを (u', v') とした際，ノード v' から出力しているエッジ中に分岐が存在していて，かつ，マッチング中のノードでないノードと接続するエッジを使用済みであった場合，マッチング中のノードを使用してしまうと，分岐であるに関わらず，同時に存在することになってしまうため不適切である．エッジが使用されているかの判定は，マッチングのために使用しようとしているエッジが，使用しているまたは使用してはいけないエッジの集合 UB 中に含まれているか調べることで行う．不適切であったと判明した場合，IsBranchFormer と同様，次のノード候補へと戻り，処理を再開することになる．

- UpdateState は IsJoinable，IsBranchFormer，IsBranch Destination をすべて適切であるという判断をなされたペア (u, v) をマッチしたペアを格納する集合 M に追加する関数である．ほかの処理として，IsBranchFormer 及び IsBranchDestination で使用したエッジ，使用したエッジと分岐していたエッジ全てを，使用しているまたは使用してはいけないエッジの集合 UB に追加する．UpdateState によって要素を追加された集合 M のサイズがパターングラフのノード数と一致した場合，本アルゴリズムは終了となる．
- RestoreState はマッチしたノードの集合 M から (u, v) を取り除く関数である．IsJoinable，IsBranchFormer，IsBranchDestination のいずれかの関数で (u, v) がマッチしないと判断された場合はこの関数を通り，次の候補にあたることとなる．

ここからは SubgraphSearch の動きを具体的に説明を進める．以下のような ShEx を考える．

$$\begin{aligned}\delta(t_0) &= a :: t_1 | a :: t_2 | b :: t_3 \\ \delta(t_1) &= b :: t_3 \\ \delta(t_2) &= c :: t_3 \\ \delta(t_3) &= \varepsilon\end{aligned}$$

上記のような ShEx を元に，前述した前処理を施して作成されるスキーマグラフは図 3 の右側のようになる．これに対する問合せとしてのパターングラフを図 3 の左側のような形とする．なお，エッジにおいて，実線で表記されたものは使用可能なもの，点線で表記されたものは使用不可なものとする．

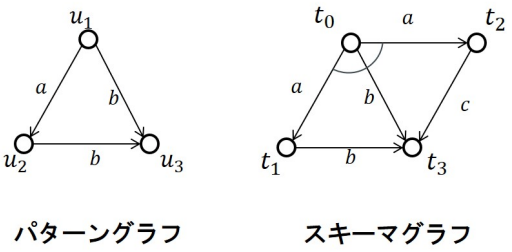

図 3　マッチング

SubgraphSearch 以前の FilterCandidates，BranchExtraction による処理によって以下の通りに集合が作成されている．

$$\begin{aligned}C(u_1) &= \{t_0\} \\ C(u_2) &= \{t_1\} \\ C(u_3) &= \{t_3\} \\ B &= \{t_0\}\end{aligned}$$

図 4 はマッチング済みのものを保存する M が空であり，(u_1, t_0) のマッチングを行っている状態を示している．集合 M は空であるため，IsJoinable を通過する．$t_0 \in B$ であるため，IsBranchFormer にて判定を行う必要があるが，集合 M は空であるため，(u_1, t_0) は不適切な組み合わせではないといえる．IsBranchDestination は条件に該当しないためスキップされる．したがって，マッチング中の (u_1, t_0) の組み合わせはマッチし，

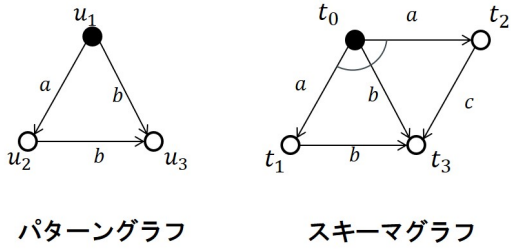

図 4　マッチング 1

集合 M に格納され，再び SubgraphSearch が呼び出される．

続いて，図 5 は $M = \{(u_1, t_0)\}$ であり，(u_2, t_1) のマッチングを行っている状態を示している．マッチング済みのノードと接続するエッジは，$u_1 \xrightarrow{a} u_2$ と $t_0 \xrightarrow{a} t_1$ であり，一致しているため，IsJoinable を通過する．$t_1 \notin B$ であるため，IsBranchFormer は条件に該当せずスキップとなるが，接続している t_0 が $t_0 \in B$ であるため，IsBranchDestination にて判定を行う必要がある．t_0 から出力している分岐をもつエッジおよびノードは，$t_0 \xrightarrow{a} t_1$，$t_0 \xrightarrow{a} t_2$，$t_0 \xrightarrow{b} t_3$ である．これらの中には使用済みのエッジを保存する集合である UB に含まれるものは存在しないため $t_0 \xrightarrow{a} t_1$ は使用してよい．マッチング中の (u_2, t_1) の組み合わせはマッチし，集合 M に格納され，集合 UB に前述の三つのエッジを保存し，再び SubgraphSearch が呼び出される．

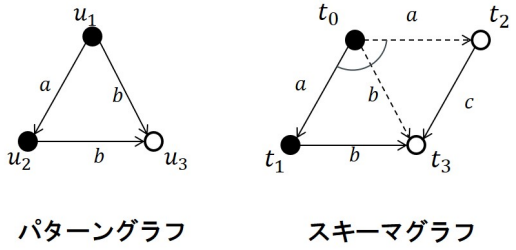

図 5　マッチング 2

最後に，図 6 は $M = \{(u_1, t_0), (u_2, t_1)\}$ であり，(u_3, t_3) のマッチングを行っている状態を示している．マッチング済みのノードと接続するエッジは，$u_1 \xrightarrow{b} u_3$ と $t_0 \xrightarrow{b} t_3$，$u_2 \xrightarrow{a} u_3$ と $t_1 \xrightarrow{b} t_3$ であり，一致しているため，IsJoinable を通過する．$t_3 \notin B$ であるため，IsBranchFormer は条件に該当せずスキップとなるが，接続している t_0 が $t_0 \in B$ であるため，IsBranchDestination にて判定を行う必要がある．t_0 から出力している分岐をもつエッジおよびノードは，$t_0 \xrightarrow{a} t_1$，$t_0 \xrightarrow{a} t_2$，$t_0 \xrightarrow{b} t_3$ である．これらの中で，$t_0 \xrightarrow{a} t_1$ は現時点までの処理の間に利用してしまっている．そのため他二本のエッジは同時に使用不可となり，$t_0 \xrightarrow{b} t_3$ を使用することができない場合，現在マッチング中の (u_3, t_3) の組み合わせは，不適当な組み合わせとなる．したがって，マッチング中の (u_3, t_3) の組み合わせはマッチ不可であり，集合 M から削除され，他候補から探索を行おうとする．しかし今回の例の場合，候補集合の $C(u_3) = \{t_3\}$ が不適であることから他に u_3 とマッチする可能性のあるノー

ドは存在しないため，充足不能と出力し終了することとなる．

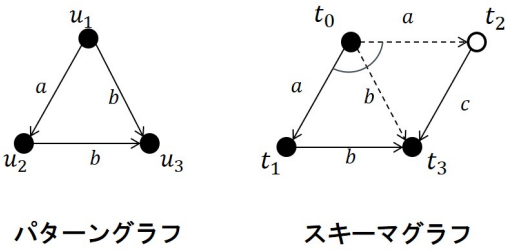

図 6　マッチング 3

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

評価実験では，始めに第 3 章で述べた提案アルゴリズムを Ruby を用いて実装した．次に，SP^2Bench で作成した RDF データと，それを元に妥当となるように作成した ShEx それぞれのグラフを用意した．最後に，作成された RDF データグラフと ShEx を元にしたグラフに充足不能なパターンを作成し，問合せにかかった時間を計測した．SP^2Bench [5] は SPARQL の包括的なベンチマークソフトであり，コンピュータ科学に関する書誌学ウェブ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の RDF データを作成する．図 7 に SP^2Bench のデータ構造，図 8 に DBLP と SP^2Bench のラベルの対応関係を示す (いずれも文献 [5] からの引用)．生成されるデータ例として評価実験に用いたファイルの一部を図 9 に挙げる．生成されるデータは，各行ごとに名前空間またはトリプルを意味する．名前空間は DBLP 固有の文書クラスを定義している．トリプルは「出力ノード ラベル 入力ノード .」とそれぞれ空白で区切られた状態で記述される．具体的には「http://localhost/publications/journals/Journal1/1940 dc:title ”Journal 1 (1940)”^^xsd:string.」といったものである．これは，http://localhost/publications/journals/Journal1/1940 という URI をもつリソースのタイトルが Journal 1 (1940) であることを表している．ノードは以下 4 種類ある．

一般ノード　リソースの URI と表記されるノード
匿名ノード　先頭に「_:」を付けて表記されるノード
リテラルノード　「”」で囲まれ，末尾に XML Schema のデータ型を表記されるノード
クラスノード　先頭に名前空間接頭辞「bench:」を付けて表記されるノード

本研究においては，生成されるデータの内，名前空間を除いたトリプルを表記した行を利用して探索を行う．また，実験ではサイズが 5,400,848B，トリプル数が 10000 のものとサイズが 16,228,701B，トリプル数が 50000 の 2 つの RDF データを作成し，利用した．

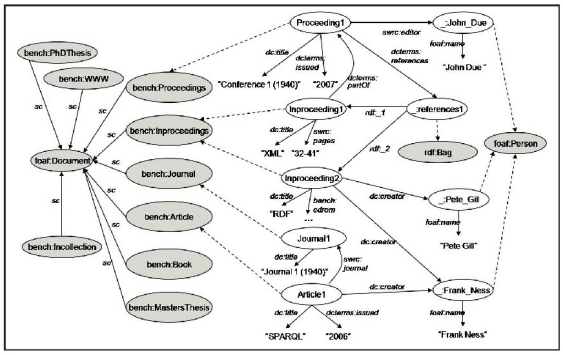

図 7　SP^2Bench のデータ構造

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^2Bench のラベルの対応関係

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix swrc: <http://swrc.ontoware.org/ontology#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix bench: <http://localhost/vocabulary/bench/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix person: <http://localhost/persons/> .
bench:Journal rdfs:subClassOf foaf:Document.
bench:Proceedings rdfs:subClassOf foaf:Document.
bench:Inproceedings rdfs:subClassOf foaf:Document.
bench:Article rdfs:subClassOf foaf:Document.
bench:Www rdfs:subClassOf foaf:Document.
bench:MastersThesis rdfs:subClassOf foaf:Document.
bench:PhDThesis rdfs:subClassOf foaf:Document.
bench:Incollection rdfs:subClassOf foaf:Document.
bench:Book rdfs:subClassOf foaf:Document.
<http://localhost/persons/Paul_Erdoes> rdf:type foaf:Person.
<http://localhost/persons/Paul_Erdoes> foaf:name "Paul Erdoes"^^xsd:string.
<http://localhost/misc/UnknownDocument> rdf:type foaf:Document.
<http://localhost/publications/journals/Journal1/1940> rdf:type bench:Journal.
<http://localhost/publications/journals/Journal1/1940> swrc:number "1"^^xsd:integer.
<http://localhost/publications/journals/Journal1/1940> dc:title "Journal 1 (1940)"^^xsd:string.
<http://localhost/publications/journals/Journal1/1940> swrc:volume "1"^^xsd:integer.
<http://localhost/publications/journals/Journal1/1940> dcterms:issued "1940"^^xsd:integer.
<http://localhost/publications/articles/Journal1/1940/Article1> rdf:type bench:Article.
```

図 9　SP^2Bench で作成されるグラフデータ例

前述したグラフ構造とラベルに基づいて ShEx を次のように定義する．

ここで $\delta(t_t)$ と $\delta(t_0)$ はどちらも ε の型であるが，t_t はリテラルノード，t_0 はクラスノードを表すため，表記を分けている．

$$
\begin{aligned}
\delta(t_t) &= \varepsilon \\
\delta(t_0) &= \varepsilon \\
\delta(t_1) &= rdf : type :: t_0 || swrc : isbn :: t_t || dc : publisher :: t_t || dc : title :: t_t || (foaf : homepage :: t_t)? || dcterms : issued :: t_t || (swrc : series :: t_t || swrc : volume :: t_t) | ((dc : creator :: t_8)^+ || (swrc : editor :: t_8)^* || (dcterms : references :: t_9)?) \\
\delta(t_2) &= rdf : type :: t_0 || bench : booktitle :: t_t || rdfs : seeAlso :: t_t || swrc : pages :: t_t || dc : title :: t_t || (foaf : homepage :: t_t)? || dcterms : issued :: t_t || (dcterms : partOf :: t_1)? || (dc : creator :: t_8)^+ || (dcterms : references :: t_9)? \\
\delta(t_3) &= rdf : type :: t_0 || swrc : number :: t_t || dc : title :: t_t || swrc : volume :: t_t || dcterms : issued :: t_t || (swrc : editor :: t_8)^* \\
\delta(t_4) &= rdf : type :: t_0 || (bench : abstract :: t_t)? || (bench : cdrom :: t_t)? || rdfs : seeAlso :: t_t || (swrc : month :: t_t)? || (swrc : note :: t_t)? || swrc : pages :: t_t || dc : title :: t_t || (foaf : homepage :: t_t)? || swrc : journal :: t3 || (dc : creator :: t8)^+ || (dcterms : references :: t9)? \\
\delta(t_5) &= rdf : type :: t_0 || dc : publisher :: t_t || dc : title :: t_t || (foaf : homepage :: t_t)? || dcterms : issued :: t_t || (dc : creator :: t_8)^+ \\
\delta(t_6) &= rdf : type :: t_0 || (rdfs : seeAlso :: t_t)? || dc : publisher :: t_t || dc : title :: t_t || (foaf : homepage :: t_t)? || dcterms : issued :: t_t || (dc : creator :: t_8)^+ \\
\delta(t_7) &= rdf : type :: t_0 || bench : booktitle :: t_t || swrc : pages :: t_t || dc : title :: t_t || (foaf : homepage :: t_t)? || dcterms : issued :: t_t || (dc : creator :: t_8)^+ || (rdfs : seeAlso :: t_t)? \\
\delta(t_8) &= rdf : type :: t_0 || foaf : name :: t_t \\
\delta(t_9) &= rdf : type :: t_0 || rdf :: : (t_2|t_3|t_4|t_6|t_7|t_8)^*
\end{aligned}
$$

また，実験には充足不能なパターンの問合せを行った場合の時間差を利用するため，図 10，図 11，図 12，図 13 の 4 つのようなパターンを作成した．

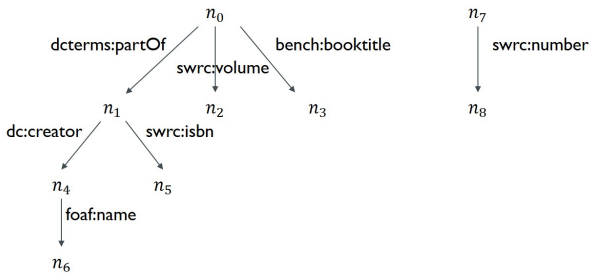

図 10　充足不能なパターン 1

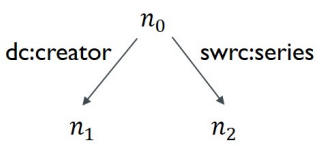

図 11　充足不能なパターン 2

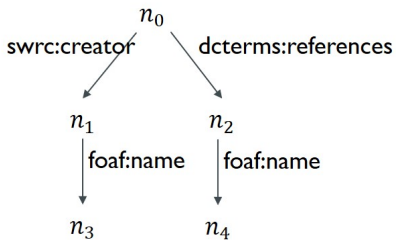

図 12　充足不能なパターン 3

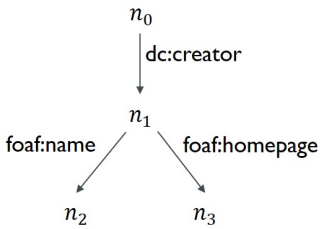

図 13　充足不能なパターン 4

評価実験の環境は以下の通りである．

プロセッサ　Intel(R) Core(TM) m3-7Y30 CPU @ 1.00GHz 1.60GHz

主記憶　4.00 GB (3.88 GB 使用可能)

OS　Windows 10 Home

使用言語　ruby 2.5.1

4.2 結　　果

前節で述べた RDF データ，ShEx，パターンを用いて充足不能なパターン問合せを行い，それぞれのグラフに対する問合せにかかる時間を計測し比較を行った．処理時間の計測は Ruby の Benchmark メソッドを使用して計測したため，測定結果の単位は秒である．結果を表 1 に示す．今回記載しているものは CPU 時間である．パターン 4 の DNF は，12 時間以内に処理が終了しなかったことを表す．時間平均はパターン 1 から 3 を用いて算出している．

表 1　実 行 結 果

	パターン 1	パターン 2	パターン 3	パターン 4	平均
提案手法	0.0000	0.0000	0.0000	0.0000	0.0000
パターン 10000	0.0940	0.0630	0.0620	DNF	0.0730
パターン 50000	0.1090	0.0470	0.0780	DNF	0.0780

4.3 考　　察

前節の結果を見ると，考案アルゴリズムによってサイズの小さな ShEx を元にしたグラフを探索させることで，パターン問合せが充足不能であった場合，パターン問合せに要する時間を大幅に短縮することが可能となる．

また，部分グラフ同型問題と同様，マッチングを行うノードの順によって実行時間に差が出ることが判明した．マッチング不能なノードがマッチングを行う順の先頭に近ければ近いほどに充足不能であると判明するまでの時間は短くなる．さ

らに，マッチング不能なことが判明し，一段階マッチングするノードを戻した場合の候補集合が大きければ大きいほど，再探索に時間がかかることも判明した．マッチング済のものが変われば充足可能になる可能性があると期待され，探索するものの，いずれの場合も充足不可とされ戻されることが影響していると考えられる．パターンのノードが少ない場合，マッチする可能性の高いものが多々存在するため，この場合も実行時間は増える．例えば，図 13 のパターンで問合せを行った場合は，$n_0 \xrightarrow{dc:creator} n_1$ は充足可能であり，RDF データグラフ中に該当する部分が多くあるが，n_1 とマッチングする可能性のあるノードは存在しないため，探索時間が長くなっていると考えられる．

5 むすび

本稿では，まず ShEx をグラフ化する手法を示し，それに基づいて ShEx の下でのパターン問合せの充足可能性判定を行うアルゴリズムを提案した．具体的には，選言「|」を用いる RBE を使用する ShEx に対するパターン問合せを可能にするために選言「|」に対応させた．また，評価実験として，RDF データグラフと ShEx を元に作成したグラフに対して，Ruby 言語を用いて実装した提案アルゴリズムを用いて実行時間を計測し，比較を行った．その結果，複数の充足不可能なパターン問合せにおいて，RDF データグラフを探索するよりも ShEx を探索する時間が短く済むことを確認した．部分グラフ同型問題と同様，探索するグラフが大きくなると実行時間は指数関数的に増加するが，充足不能性の判定を行うことでパターン問合せの実行時間にほぼ影響を与えずに充足可能性判定を行えることも確認した．

今後の課題として，本稿では single type semantics でのみ検討したため，multi type semansics にも対応させること，単一のパターングラフのラベル以外にも対応させること，アルゴリズムの改良が挙げられる．アルゴリズムの改良の具体案としては，部分グラフ同型問題にもある通り，高速化や効率化がある．今後は，これらの課題を解決したよりよいアルゴリズムを考案予定である．

文　献

[1] World Wide Web Consortium (W3C). "RDF Schema 1.1". https://www.w3.org/TR/rdf-schema/. (accessed 2018-12-02).

[2] World Wide Web Consortium (W3C). "Shape Expressions (ShEx) Primer". http://shex.io/shex-primer/. (accessed 2018-12-02).

[3]] J. R. Ullmann. An algorithm for subgraph isomorphism. J.ACM, 23:3142, January 1976.

[4] Jinsoo Lee, Romans Kasperovics, Wook-Shin Han, Jeong-Hoon Lee. "An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases". Proceedings of the VLDB Endowment. 2012, 6, 2, 133-144.

[5] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel, "SP2Bench: a SPARQL Performance Benchmark," Proc. ICDE, 2009, pp. 371-393.