

Shape Expression Schema の下での Conjunctive
Property Path 充足可能性判定手法

筑波大学

人間総合科学学術院人間総合科学研究群

情報学学位プログラム

2022年03月

前田 祐紀

Shape Expression Schema の下での Conjunctive Property Path 充足可能性判定手法

A Method for Determining Conjunctive Property Path Satisfiability under Shape Expression Schema

氏名：前田 祐紀
Maeda Yuuki

近年、RDF データ (グラフデータ) の社会への普及が進んでいる。しかし、グラフデータの構造を示すためのスキーマの活用は未だ途上の域にあり、既存スキーマはいずれもスキーマ言語としての表現力が不十分である。そこで、従来のスキーマよりも適切にグラフデータの構造を記述することが可能な Shape Expression Schema (ShEx) が W3C Draft Community Group によって提案されている。問合せ P と ShEx S に対して、 S に妥当などのグラフデータにおいても P の解が空である場合、 P は充足不能であるという。充足不能な問合せの結果は必ず空であり、問合せの実行は無駄となるため、充足不能な問合せを実行前に検出することが望ましい。

そこで本研究では、ShEx の下での問合せ充足可能性問題について考察する。本研究は問合せとして Property Path を一般化した Conjunctive Property Path (CPP) を対象としている。充足可能性判定問題ではこれまでに DTD や XML スキーマの下での XPath 充足可能性判定、ShEx の下でのパターン問合せ充足可能性判定等が研究されている。しかし、著者の知る限りで ShEx の下での CPP 充足可能性判定に関する研究は行われていない。本研究の目的が達成されることで、大規模なグラフデータに対してより効率の良い問合せが可能となり、グラフデータの利用者にとって有用であると考えられる。

提案手法では、CPP 問合せ Q を有向グラフ G_Q で表し、各 Property Path について充足可能性判定を行いそれぞれの解を順に求め、各判定で得られた解 (型) をノード候補として保持し、解の不動点計算を行う。すなわち、 G_Q 上で同じノードに複数の解が得られた場合はそれらの共通集合を取り、新たな Property Path を判定する毎にノード候補の更新を行う。全ての Property Path の判定後に条件を満たすノード組が存在すれば充足可能、なければ充足不能と判定する。CPP 充足可能性判定のために Property Path の充足可能性判定アルゴリズムを併せて作成した。このアルゴリズムはオートマトンに基づいて構成されている。まず、Property Path をオートマトン M_P に、ShEx をオートマトン M_G に変換する。そして M_G 上に M_P を満たす経路があれば充足可能、一つもなければ充足不能と判定できる。

評価実験では、充足不能な CPP 問合せを生成し、それらの問合せに要する時間、および、提案アルゴリズムによる充足可能性判定を行なった。その結果、グラフデータの問合せ時間と比較して、充足可能性判定に要する時間が十分に小さいことを確認した。

主研究指導教員：鈴木 伸崇
副研究指導教員：阪口 哲男

目次

第 1 章 序章	1
第 2 章 諸定義	3
2.1 グラフ	3
2.2 Shape Expression Schema	3
2.3 Property Path	5
2.4 Conjunctive Property Path	6
2.5 充足可能性問題	6
第 3 章 提案手法	7
3.1 Property Path の充足可能性判定	7
3.2 CPP の充足可能性判定	15
3.3 アルゴリズムの正当性	17
第 4 章 評価実験	18
4.1 概要	18
4.2 結果	22
4.3 考察	23
第 5 章 まとめ	24
謝辞	25
参考文献	26

目次

2.1.1 グラフ G_0	3
2.2.1 ShEx の例	4
2.2.2 グラフ G	5
3.1.1 Property Path, ShEx から変換したオートマトン	11
3.1.2 マッチング 1-1	12
3.1.3 マッチング 1-2	12
3.1.4 マッチング 2-1	13
3.1.5 マッチング 2-2	13
3.1.6 マッチング 2-3	14
3.1.7 マッチング 3-1	14
4.1.1 SP ² Bench のデータ構造	19
4.1.2 DBLP と SP ² Bench のラベルの対応関係	19
4.1.3 BSBM のデータ構造	21

第1章 序章

近年, RDF データ (グラフデータ) の社会への普及が進んでいる. しかし, グラフデータの構造を示すためのスキーマの活用は未だ途上の域にある. グラフデータに対するスキーマとしては, これまでグラフスキーマなどが提案されているが, いずれも繰り返しや選言などの指定ができず, 表現力が不十分である. また, RDF データのスキーマ言語として RDF Schema (RDFS) [1] が提案されているが, RDFS はオントロジー定義言語としての性質が強く, スキーマ定義には必ずしも適していない. そこで, 従来のスキーマよりも適切にグラフデータの構造を記述することが可能な Shape Expression Schema (以下 ShEx) [2] が提案されている. オントロジーのセマンティクスとは異なり, ShEx はグラフデータの構造的特徴を捉えられるように設計されており, すでに様々な分野で使用されている [3]. ShEx はノードに対する型を Regular Bag Expression という規則に基づいて定義することが特徴のスキーマ言語である.

本研究では, ShEx における問合せの充足可能性判定について考える. 問合せとしては, SPARQL 1.1 で定義されている Property Path [4] を一般化した Conjunctive Property Path (以下 CPP) を対象とする. ここで, 問合せ P と ShEx S に対して, S に妥当などのグラフデータにおいても P の解が空である場合, P は充足不能であるという. 問合せ P が与えられた場合, P を用いてグラフデータを探索する必要がある. しかし, 一般的にグラフデータのサイズは非常に大きい. したがって, P が充足不能な場合, 莫大な量のデータに対してそのような探索を行うのは明らかに非効率的である. 一方, ShEx はグラフデータよりサイズが著しく小さく, 充足可能性判定は効率よく行えると期待される. 問合せの実行に先立ってその充足可能性判定を行うことにより, 充足不能な問合せの実行を回避することができる.

本研究では, グラフデータに比べてサイズの小さい ShEx から CPP 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. 具体的には, まず CPP Q を有向グラフ G_Q に変換する. 次に ShEx と各 Property Path をそれぞれオートマトン M_S, M_P に変換する. オートマトンとは状態と遷移の集合によって自動機械の仕組みを表すモデルである. 各 Property Path について, M_S の各ノードを M_P の開始状態と見立てて M_P の受理状態まで遷移できるかを調べる. 一つでも条件を満たすパスが見つければ充足可能, 見つからなければ充足不能と判定することができる. G_Q のエッジに沿って各 Property Path についてこの判定を行い, 解として得られたノード候補を保持していく. 判定を進めながらその都度ノード候補を更新する手順を全ての Property Path の判定が終わるまで繰り返すことで Q を満たす解が存在するかを判定する.

以上のアルゴリズムを実装し, 充足不能な CPP 問合せを用いて評価実験を行なった. その結果, 提案アルゴリズムで充足可能性判定を行うことができ, 充足可能性判定に要する時間はグラフデータの問合せ時間と比較して大幅に小さいことを確認した.

問合せの充足可能性判定問題はデータベース分野における主要な問題である. DTD や XML スキーマの下での XPath 充足可能性判定に関する研究は既に行われている [5, 6]. しかし, XML が順序付き木であるのに対しグラフデータは非順序付きグラフであるため, これらの研究は RDF へ適応させることができない. また, スキーマを用いずにパターン問合せ

の充足可能性判定を考える研究 [7] や, ShEx の下でパターン問合せを用いて充足可能性判定を行うアルゴリズムの提案がある [8]. パターン問合せは Property Path と同様に SPARQL 1.1 で定義されている, 探索したいグラフの形 (パターン) を指定してそのパターンにマッチする部分をグラフデータから探索する問合せ方法である. 本研究の対象である Property Path では, 探索したい経路に対する条件を記述し, 指定した経路の開始ノードと終端ノードのペアが解となる. 例として, 「あるラベルが 0 回以上出現する」という条件に対しては指定のラベルが何回出現しても, もしくは出現しなくてもマッチするということになる. さらに, パターン問合せは CPP 問合せの部分集合であると言える. したがって, 両者は互いに異なるものであり, 充足可能性の判定には異なる手法が必要となる. ShEx の下で CPP 問合せの充足可能性問題を解くアルゴリズムは著者の知る限り提案されていない.

本論文の構成は以下の通りである. 第 2 章では, グラフ, ShEx, Property Path, Conjunctive Property Path, 充足可能性問題の定義を述べる. 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, 本研究のまとめを述べる.

第2章 諸定義

本章では、本論文で扱うグラフ, Shape Expression Schema, Property Path, Conjunctive Property Path, 充足可能性問題について述べる.

2.1 グラフ

本研究におけるグラフはラベル付き有向グラフを用いる. Σ 上のグラフ G を, $G = (V, E)$ として定義する. 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 2.1.1 にグラフ G_0 を示す.

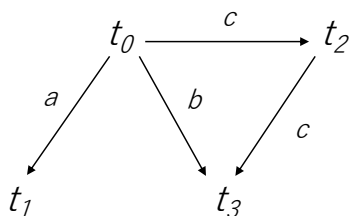


図 2.1.1: グラフ G_0

定義に基づいて上記のグラフ G_0 を表記すると,

$$\begin{aligned}\Sigma &= \{a, b, c\}, \\ V &= \{t_0, t_1, t_2, t_3\}, \\ E &= \{(t_0, a, t_1), (t_0, b, t_3), (t_0, c, t_2), (t_2, c, t_3)\}\end{aligned}$$

となる. また, グラフ内のある辺において, 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.

2.2 Shape Expression Schema

本節では, Shape Expression Schema (以下 ShEx) の定義を述べる.

ShEx はグラフデータに対するスキーマ言語の一つである. スキーマ言語とは XML における DTD や XML Schema のように, グラフデータの属性などの構造を示すための言語である. ShEx では, ノードに対する型の定義について Regular Bag Expression (以下 RBE) [9] という規則を用いている. 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. RBE の規則は以下のようにになっている.

- ε 及び任意の $a :: t \in \Sigma \times \Gamma$ は RBE である.

- r_1, r_2, \dots, r_k を RBE とするとき, $r_1 || r_2 || \dots || r_k$ は RBE である. ここで, ‘||’ は順序を無視した連結を表す.
- r_1, r_2, \dots, r_k を RBE とするとき, $r_1 | r_2 | \dots | r_k$ は RBE である. ここで, ‘|’ は選言を表す.
- r を RBE とするとき, $r^{[n,m]}$ は RBE である. ここで, $[n,m]$ は $n \leq m$ であり, n 回以上 m 回以下の繰り返しであることを表す. 特に, $r^? = r^{[0,1]}$, $r^+ = r^{[1,\infty]}$, $r^* = r^{[0,\infty]}$ である.

例として, $r = (a :: t_1 | b :: t_2) || c :: t_3$ という RBE を考える. ‘||’ は順序を無視した連結であるため, r は $a :: t_1, c :: t_3$ と $b :: t_2, c :: t_3$ に加えて $c :: t_3, a :: t_1$ と $c :: t_3, b :: t_2$ を含む.

ShEx は $S = (\Sigma, \Gamma, \delta)$ と定義される. ここで, Σ はラベルの集合, Γ は型名の集合, δ は RBE に基づいた型を定義する関数である. 図 2.2.1 に示した ShEx は以下の $S = (\Sigma, \Gamma, \delta)$ で表される.

$$\begin{aligned}
\Sigma &= \{\text{takes, supervisor, teaches}\}, \\
\Gamma &= \{t_1, t_2, t_3\}, \\
\delta(t_1) &= (\text{takes} :: t_2)^* || (\text{supervisor} :: t_3)^?, \\
\delta(t_2) &= \varepsilon, \\
\delta(t_3) &= (\text{teaches} :: t_2)^*
\end{aligned}$$

$$\begin{aligned}
&< t_1 > \{ \\
&\quad \text{takes}@ < t_2 >^* || \\
&\quad \text{supervisor}@ < t_3 >^? \\
&\} \\
&< t_2 > \{ \\
&\} \\
&< t_3 > \{ \\
&\quad \text{teaches}@ < t_2 >^* \\
&\}
\end{aligned}$$

図 2.2.1: ShEx の例

ここで, S と図 2.2.2 に示したグラフ G を考える. RBE ではエッジ e が a とラベル付けされ, かつ, e の入力ノードの型が t の場合, $a :: t$ は e と一致する. したがって, 各エッジについて比較すると G が S の有効なグラフであることを確認できる.

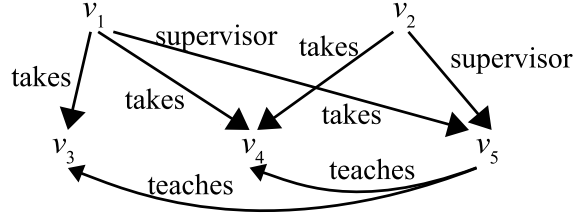


図 2.2.2: グラフ G

ShEx は二つのセマンティクスを持ち、それぞれ single type semantics (以下 s-typing), multi type semantics (以下 m-typing) と呼ばれる. $\lambda : V \rightarrow \Gamma$ は全てのノード $v \in V$ を $\lambda(v)$ に結びつける. このとき,

$$\text{out-lab-type}_G^\lambda(v) = \{ | a :: \lambda(v) \mid (v, a, v') \in E \},$$

ここで $\{ | \dots \}$ は Bag を表す. 各ノード $v \in V$ に対して $\text{out-lab-type}_G^\lambda(v)$ が $L(\delta(\lambda(v)))$ に含まれる場合 s-typing となる. m-typing では $\lambda : V \rightarrow 2^\Gamma$ となり, 各ノード $v \in V$ を型集合 $\lambda(v)$ に結びつける. ここで $\text{out-lab-type}_G^\lambda(v) = L(\text{Flatten}(\text{out-lab-type}_G^\lambda(v)))$ とする. Flatten 関数はラベルとノードの組合せを平坦化するものであり, 使用例として,

$$\text{Flatten}(\{ | a :: \{t_0, t_1\}, a :: \{t_0, t_1\}, b :: t_1 \}) = (a :: t_0 | a :: t_1) || (a :: t_0 | a :: t_1) || (b :: t_1)$$

があげられる. もし $\text{out-lab-type}_G^\lambda(v) = \{ | a :: \{t_1, t_2\}, b :: \{t_3\} \}$ であれば, $\text{out-lab-type}_G^\lambda(v) = \{ \{ | a :: t_1, b :: t_3 \}, \{ | a :: t_2, b :: t_3 \} \}$ となる. 以下の条件を満たす λ が存在するとき, G は m-typing の下で妥当である.

1. 全ての $v \in V$ について $\lambda(v) \neq \emptyset$ であり, かつ,
2. 全ての $v \in V$ と全ての $t \in \lambda(v)$ について, $\text{out-lab-type}_G^\lambda(v) \cap \delta(t) \neq \emptyset$

2.3 Property Path

本節では, Property Path の定義を述べる.

Property Path は RDF データに対する問合せの一種である. Property Path の仕様は SPARQL 1.1 の仕様の中に含まれており, XML の XPath のように探索したい経路に対する条件を記述する.

Property Path は正規表現とほぼ同様の定義がされる. ただし, いくつかの機能によって拡張されている. Σ をラベルの集合としたとき, Σ 上での Property Path の定義は以下のようになっている.

- ε および任意の $a \in \Sigma$ は Property Path である.
- $*$ はワイルドカードを表す Property Path である.
- $\{a_1, \dots, a_k\}$ をラベル集合としたとき, $!\{a_1, \dots, a_k\}$ は a_1, \dots, a_k 以外のラベルにマッチする Property Path である.
- 任意の $a \in \Sigma$ に対して, a^{-1} はラベル a のエッジを逆向きに辿る Property Path である.

- r_1, \dots, r_k を Property Path とするとき, $r_1.r_2.\dots.r_k$ は Property Path である. ここで, \cdot は連結を表す.
- r_1, \dots, r_k を Property Path とするとき, $r_1|r_2|\dots|r_k$ は Property Path である. ここで, $|$ は選言を表す.
- r を Property Path とするとき, $r^{[n,m]}$ は Property Path である. ここで, $[n, m]$ は $n \leq m$ であり, n 回以上 m 回以下の繰り返しであることを表す. 特に, $r^? = r^{[0,1]}$, $r^+ = r^{[1,\infty]}$, $r^* = r^{[0,\infty]}$ である.

例として, $\text{supervisor.teaches}^?$ や $\text{takes}^+.\text{supervisor}$ は 2.2 節の ShEx S 上の Property Path である.

2.4 Conjunctive Property Path

本節では, Property Path の拡張である Conjunctive Property Path の定義を述べる.

Conjunctive Property Path 問合せ (以下 CPP 問合せ) は変数を用いて結合演算子で繋いだ Property Path によって構成される. CPP 問合せ Q は以下の式で表される.

$$Q = \wedge_i(x_i, p_i, y_i)$$

ここで, x_i, y_i は変数, p_i は Property Path である. グラフ G 上において (x_i, p_i, y_i) が真と評価されるのは, ある割り当て $x_i \leftarrow v$ と $y_i \leftarrow v'$ に対して, パス上のラベルが p_i を満たすような v から v' へのパスが G に含まれている場合である.

また, Q に現れる変数の集合を $\text{Var}(Q)$ とし, 各変数と各 Property Path の組 (x_i, p_i, y_i) をそれぞれノードと “ p_i ” でラベル付けした x_i から y_i への辺とみなして得られるグラフを $G(Q)$ とする. 問合せ (または $G(Q)$ 上のエッジ) (x_i, p_i, y_i) において x_i を head, y_i を tail と呼称する.

2.5 充足可能性問題

本節では, 充足可能性問題を定義する.

Property Path P に対して, グラフ上の経路でそのラベル列が P を満たすものであれば, その経路の開始ノードと終端ノードのペアが P の解となる. Property Path P と ShEx S に対して, 「 S に妥当なあるグラフデータに対して空でない P の解が存在する」ときに P は S の下で充足可能であるという. 逆に, S に妥当などのグラフデータに対しても S の解が空となるとき, P は S の下で充足不能であるという.

例えば, 2.2 節の S において 2.3 節で挙げた Property Path を考えると, $\text{supervisor.teaches}^?$ は S の下で充足可能であり, 一方 $\text{takes}^+.\text{supervisor}$ は S の下で充足不能である.

さらに, 本研究では問合せとして CPP を用いている. CPP 問合せ Q と ShEx S があるとき, S に妥当なあるグラフデータ G に対して, Q が G 上で真と評価される $\text{Var}(Q)$ の割り当てがある場合に, Q は S の下で充足可能という. このように, 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.

第3章 提案手法

この章では、ShExにおけるCPP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 Property Pathの充足可能性判定

本節では、ShExにおけるProperty Path問合せの充足可能性判定アルゴリズムに関して述べる。

提案アルゴリズムでは、オートマトンを利用する。オートマトンは、状態とその遷移関数の集合によって自動機械の仕組みを表すモデルである。その定義を以下に示す。

- オートマトン M は $M = (Q, \Sigma, \delta, q_I, F)$ によって表される。
- Q : 状態の集合
- Σ : ラベルの集合
- $\delta : Q \times \Sigma \rightarrow Q$ (遷移関数)
- q_I : 開始状態
- F : 受理状態

まず、ShExをオートマトン M_G に変換する。ただし、スキーマは開始状態と受理状態という概念を持たないため、 M_G には q_I, F が存在しない。ShExの定義に従って変換を行うと以下のように表せる。

- ShEx $S = (\Sigma_1, \Gamma_1, \delta_1)$ からオートマトン $M_G = (Q_2, \Sigma_2, \delta_2, nil, \emptyset)$ に変換する。ここで、
- $Q_2 = \Gamma_1$
- $\Sigma_2 = \Sigma_1$
- $\delta_2(t, a) = \{t' \mid a :: t' \text{ が } \delta_1(t) \text{ に出現する}\}$

ここで、Property Pathは条件を満たす経路を求める問合せであるため、型の定義におけるエッジの出現回数、及び選言を考慮する必要がない。すなわち、0回または1回の出現を表す '?', 1回以上の出現を表す '+', 0回以上の出現を表す '*' はそれぞれ1回の出現とする。さらに、型の定義内で選言を含むノードを一度通ったとしても再び到達するのは同じ型の異なるノードとみなせるため、選言を表す '|' は順序を無視した連結を表現する '||' と同じとみなしてよい。

次に、Property Pathで記述された問合せをオートマトン M_P に変換する。Property Pathは正規表現を含んでいるが、正規表現からオートマトンへの変換方法は既に存在しており [10], これを利用している。

ここまでの処理ののち、 M_G が M_P を含んでいるかを調べる。提案アルゴリズムを Algorithm 3.1.1 に示す。また、本アルゴリズムでは Property Path において1つのラベルに!と-1 が同時に出現することはないと仮定している。

Algorithm 3.1.1 PP SATISFIABILITY DECISION

Input: ShEx schema $S = (\Sigma, \Gamma, \delta)$, Property Path P

Output: “satisfiable” or “unsatisfiable”

```

1:  $M_G := \text{ShExToAutomaton}(S)$ ;
2:  $M_P := \text{PpToAutomaton}(P)$ ;
3:  $\Sigma' := \Sigma(M_G) \cup \Sigma(M_P)$ ;
4: for each  $n \in Q(M_G)$  do
5:    $curstate := [0, n]$ ;
6:    $arrives \leftarrow curstate$ ;
7:    $\text{TransCheck}(M_G, M_P, curstate, arrives, \Sigma')$ ;
8: end for
9: report “unsatisfiable”;

```

Algorithm 3.1.2 TransCheck

Input: $M_G, M_P, curstate, arrives, \Sigma'$

```

1: for each  $sn \in \Sigma(M_P)$  do
2:   if  $sn[0] = '!$  then
3:      $Excl := \text{MakeExclamation}(M_P, curstate)$ ;
4:     for each  $sm \in \Sigma'$  do
5:       if  $sm \notin Excl$  then
6:          $\text{ExclTransNode}(M_G, M_P, sm, sn, curstate, arrives, \Sigma')$ ;
7:       end if
8:     end for
9:   else
10:    if  $sn[-1] = '-'$  then
11:       $\text{MinusTransNode}(M_G, M_P, sn, curstate, arrives, \Sigma')$ ;
12:    else
13:       $\text{TransNode}(M_G, M_P, sn, curstate, arrives, \Sigma')$ ;
14:    end if
15:  end if
16: end for

```

Algorithm 3.1.3 TransNode

Input: $M_G, M_P, sn, curstate, arrives, \Sigma'$

```

1:  $ppstate := \delta(M_P)(curstate[0], sn)$ ;
2: for each  $shexstate \in \delta(M_G)(curstate[1], sn)$  do
3:    $curstate := [ppstate, shexstate]$ ;
4:    $\text{ArriveCheck}(M_G, M_P, curstate, arrives, \Sigma')$ ;
5: end for

```

Algorithm 3.1.4 ArriveCheck

Input: $M_G, M_P, curstate, arrives, \Sigma'$

```
1: if  $\forall s \in curstate (s \neq nil \wedge s \geq 0)$  then
2:   if  $curstate[0] \in F(M_P)$  then
3:     report “satisfiable”;
4:   else
5:     if  $curstate \notin arrives$  then
6:        $arrives \leftarrow curstate$ ;
7:       TransCheck( $M_G, M_P, curstate, arrives, \Sigma'$ );
8:     end if
9:   end if
10: end if
```

まず、1~2行目で ShEx で記述されたグラフスキーマと Property Path で記述された問合せをオートマトン M_G, M_P に変換する。ここで変換のために呼び出す関数について説明する。

- ShExToAutomaton は ShEx で記述されたグラフスキーマをオートマトンに変換する関数である。本節冒頭で述べた通り、グラフスキーマはオートマトンの定義のうち開始状態と受理状態の概念を持たず、残る集合は ShEx の定義から変換可能である。
- PpToAutomaton は Property Path をオートマトンに変換する関数である。前章の 2.3 節にある通り Property Path は正規表現とほぼ同様の定義がされるがいくつかの機能によって拡張されている。正規表現をオートマトンに変換するアルゴリズムは既に考案されており、正規表現部分についてはこれを利用して変換を行なっている。また、正規表現にはない特殊記号についてもこの関数の中で処理を行なっている。-1 は該当するラベルに「-」をつけ、! は該当するラベルに「!」をつける。例えば、 a^{-1} というラベルは $a-$ に、 $\{!b\}$ というラベルは $!b$ に変換している。つまり、Property Path のオートマトンにおいて、-1 に関してはラベル a のエッジを逆向きに辿るのではなく、「ラベル $a-$ で伸びるエッジを辿る」とする。同様にして、! に関しては b 以外のラベルで辿るのではなく、「ラベル $!b$ で伸びるエッジを辿る」とする。ここでラベルに加えた「-」や「!」を目印にして充足可能性判定時の処理を変えていく。

以下、 M_G のラベル集合、状態集合、遷移関数をそれぞれ $\Sigma(M_G), Q(M_G), \delta(M_G)$ と表す。 M_P も同様の表記を用いる。

3行目で M_G, M_P のラベルをまとめた集合 Σ' を用意する。4行目からは M_G の各ノード n に対して n を Property Path の開始状態と見立てて TransCheck 関数を再帰的に呼び出す。 $curstate$ は現在いるノードの位置を示すための配列であり、0番目には M_P のノードの位置、1番目には M_G のノードの位置が格納される。 $arrives$ は既に出現したノード組を格納するための配列であり、 $curstate$ をループ開始時に格納している。

Algorithm 3.1.2 に示した TransCheck 関数では M_P の各ラベル sn で現在いるノードからの遷移を調べる。 sn の値に対して以下のように処理を行う。

- ラベルに「!」も「-」も含まれていない場合は TransNode 関数を呼び出して $curstate$ を遷移させる。TransNode 関数は Algorithm 3.1.3 に示している。これは、指定したラ

ベル sn で現在いるノードから遷移させる関数である。遷移後に ArriveCheck 関数を呼び出している。

- 「!」がある場合は MakeExclamation 関数を用いて調べる対象から外すラベル集合 $Excl$ を求める。MakeExclamation は M_P の現在のノードから伸びるエッジのうち「!」付きのラベル集合を求める関数である。例えば、現在のノードからラベル「! $\{a|b\}$ 」が伸びている Property Path の場合には a, b が格納される。これは例で挙げたように「!」で指定されるラベルが1つではなく「|」を用いて複数存在する場合があります、それらは全て選択しないようにする必要があるためである。次に Σ' の各ラベル sm について、 $Excl$ に含まれていない場合に ExclTransNode 関数を呼び出して $curstate$ を遷移させる。その際、 M_G の遷移ラベルは sm 、 M_P の遷移ラベルは sn となる。遷移後は TransNode 関数と同様に ArriveCheck 関数を呼び出している。
- 「-」がある場合は MinusTransNode 関数を呼び出して $curstate$ を遷移させる。MinusTransNode は指定したラベル sn で $\Gamma(M_G)$ の各ノードから遷移できるかを調べ、遷移できる場合はそのノードを M_G の遷移後の位置として遷移させる関数である。遷移後は TransNode 関数と同様に ArriveCheck 関数を呼び出している。

それぞれの関数を用いて $curstate$ を遷移させた後はいずれも ArriveCheck 関数を呼び出している。ArriveCheck を Algorithm 3.1.4 に示す。これは与えられた Property Path が充足可能かの判定と M_G, M_P のノード組が既に到達したものであるかを判定する関数である。まず、 $curstate$ の各要素 s 、すなわち M_G, M_P の現在いるノードが有効なノードであるかを確認する。変換後のオートマトンはどちらもノードを 0 以上の整数で表し、遷移できないラベルでの遷移先には -1 を格納している。つまり、全ての s が null でない、かつ負でない時に $curstate$ が有効として処理を進める。その後、現在の M_P のノードが M_P の受理状態の集合に含まれていれば充足可能と出力して処理を終了する。含まれていない場合のうち、 $curstate$ が $arrives$ に含まれていなければ $curstate$ を $arrives$ に格納し、再度 TransCheck 関数を呼び出す。 $curstate$ が一度到達したノード組である場合、そこから遷移できるノードは一度目に到達した時と全く同じになるので遷移処理を進める必要がないためである。

上記の処理を繰り返していき、いずれの遷移ルートも条件を満たさない場合には充足不能と出力する。

ここからは、例を用いて TransCheck の具体的な動きを説明する。以下のような Property Path と ShEx を考える。

$$\begin{array}{l|l}
 & \Sigma = \{a, b, c\} \\
 & \Gamma = \{t_0, t_1, t_2, t_3\} \\
 & \delta(t_0) = (a :: t_1 | c :: t_2)^* \\
 (b|(c.a)).c^{-1}.b & \delta(t_1) = b :: t_3 | c :: t_4 \\
 & \delta(t_2) = c :: t_3 \\
 & \delta(t_3) = b :: t_0 \\
 & \delta(t_4) = a :: t_3 \\
 \text{Property Path} & \text{ShEx}
 \end{array}$$

上記の ShEx と Property Path から ShExToAutomaton と PpToAutomaton の処理を通して図 3.1.1 のようなオートマトンを作成する。これらをそれぞれ M_G, M_P とする。 M_G, M_P

は具体的には以下の情報が格納されている。ここでは遷移先がないラベルでの遷移関数は表記していないが、実際には「-1」に遷移するという形で格納されている。

$Q = \{u_0, u_1, u_2, u_3, u_4\}$ $\Sigma = \{a, b, c, c-\}$ $\delta = (u_0, c, u_2), (u_0, b, u_1), (u_2, a, u_1),$ $(u_1, c-, u_3), (u_3, b, u_4)$ $q_I = \{u_0\}$ $F = \{u_4\}$ <p style="text-align: center;">Property Path</p>	$Q = \{t_0, t_1, t_2, t_3\}$ $\Sigma = \{a, b, c\}$ $\delta = (t_0, a, t_1), (t_0, c, t_2), (t_1, b, t_3), (t_1, c, t_4),$ $(t_2, c, t_3), (t_3, b, t_0), (t_4, a, t_3)$ <p style="text-align: center;">ShEx</p>
---	--

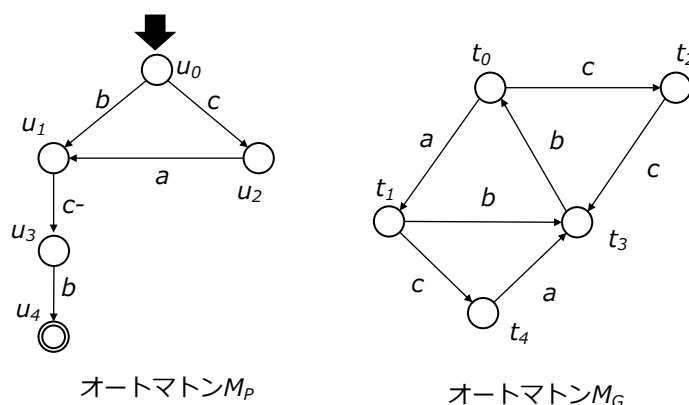


図 3.1.1: Property Path, ShEx から変換したオートマトン

最初に、 $curstate = [u_0, t_0]$ として処理を開始する。この時に $arrives$ に $[u_0, t_0]$ が格納される。図 3.1.2 は $curstate$ の位置に印をつけたものである。ラベルを b として遷移させると、 $curstate = [u_1, -1]$ となる。この $curstate$ には遷移できていないことを示す -1 が格納されているため、ArriveCheck が呼び出された際に無効と判定され処理が止まる。次に、ラベルを c として遷移させると、 $curstate = [u_2, t_2]$ となる。この $curstate$ は正常に遷移できているため ArriveCheck で有効と判定される。 u_2 は M_P の受理状態に含まれていないため、充足可能と判定されず処理が続けられる。また、 $[u_2, t_2]$ は $arrives$ にも含まれていないため $arrives$ に格納し、これを $curstate$ として再度 TransCheck を呼び出す。

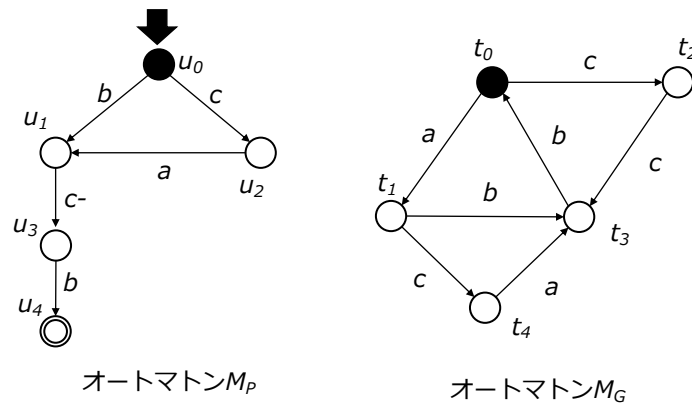


図 3.1.2: マッチング 1-1

図 3.1.3 は $curstate = [u_2, t_2]$ の状態の図である。前述の手順と同様に進めるが、 u_2 から遷移できるラベルは a のみなのに対して t_2 は a では遷移できないため、ここで処理が止まる。これで開始状態を t_0 とした時の遷移ルートは全て調べたので、別のノードを開始状態として同様の処理を行っていく。

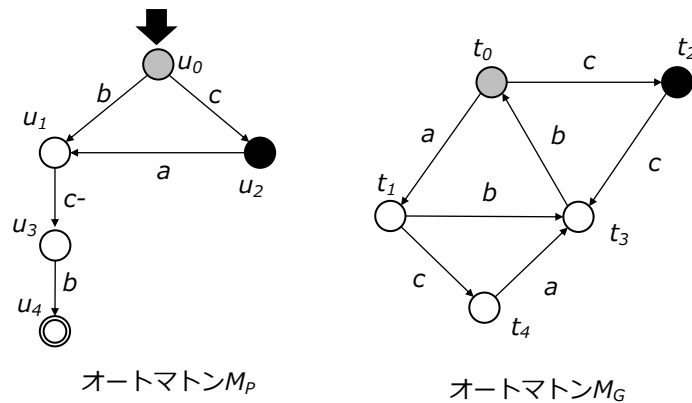


図 3.1.3: マッチング 1-2

次に、 $curstate = [u_0, t_1]$ として処理を開始する。 $arrives$ は開始状態のノードが切り替わる際に初期化されるため、先程到達したノード組は含まれていない。図 3.1.4 は $curstate$ を切り替えて $arrives$ を初期化した後の図である。ラベルを b として遷移させると $curstate = [u_1, t_3]$ となり、 u_1 が受理状態でなく $curstate$ が $arrives$ にも含まれていないため $arrives$ に追加したのち $curstate = [u_1, t_3]$ として TransCheck を呼び出す。

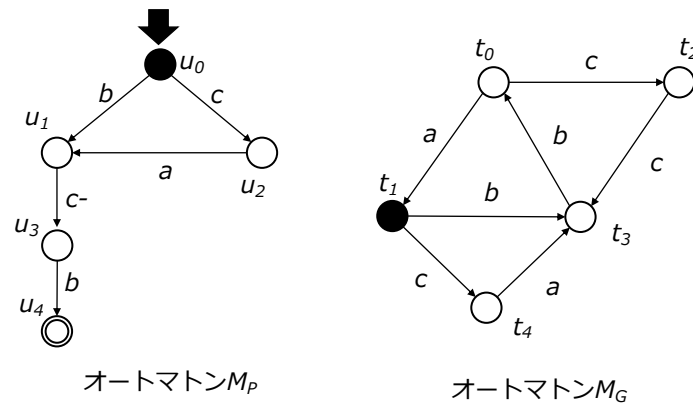


図 3.1.4: マッチング 2-1

図 3.1.5 は $curstate = [u_1, t_3]$ の状態を表した図である．次はラベル $c-$ のため Minus-TransNode を呼び出す．具体的には M_G の各ノードからラベル c で t_3 に遷移できるノードがあるかを調べている．ここでは t_2 が条件を満たすため，遷移後は $curstate = [u_3, t_2]$ として ArriveCheck を呼び出す．この例では遷移先は 1 つだが，該当するノードが複数ある場合も全ての $curstate$ で ArriveCheck を呼び出す． u_3 が受理状態に含まれておらず， $[u_3, t_2]$ も *arrives* に含まれていないため，ノード組を格納して TransCheck を呼び出す．

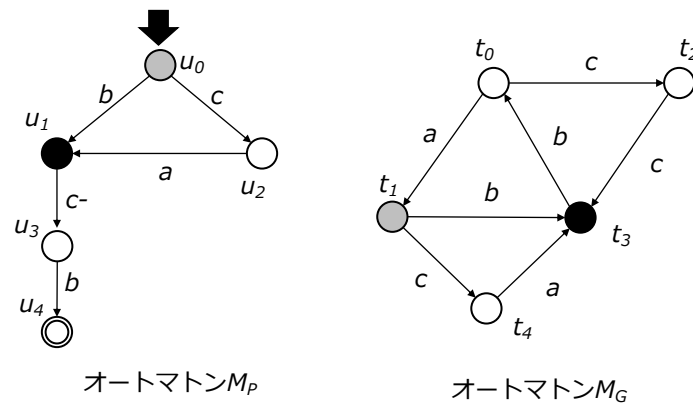


図 3.1.5: マッチング 2-2

図 3.1.6 は $curstate = [u_3, t_2]$ の状態であるが， u_3 から遷移するためのラベル b では t_3 から遷移できないためここで処理が止まる．

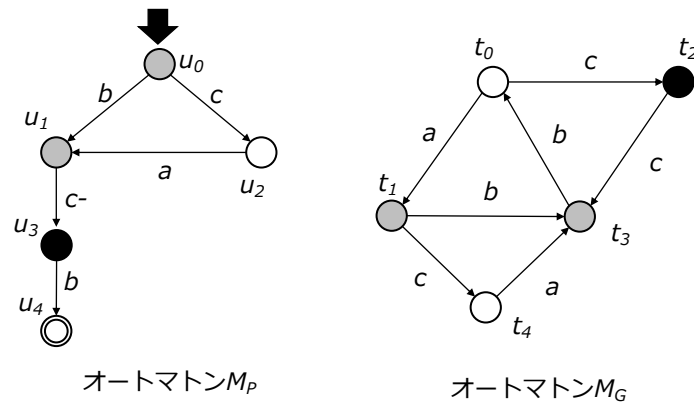


図 3.1.6: マッチング 2-3

次は $curstate = [u_0, t_1]$ まで戻り、ラベル c での遷移を調べていく処理を行う。同様の処理を施すと、ラベル c で $[u_2, t_4]$ 、その後ラベル a で $[u_1, t_3]$ まで遷移することができる。ここで、 $[u_1, t_3]$ は *arrives* に格納されている。すなわち、既に到達したノード組であり、ここから遷移できる可能性のあるルートは全て探索済みであるため、この時点で処理を終了する。これで t_1 を開始状態とした時の遷移ルートは全て調べたと言える。

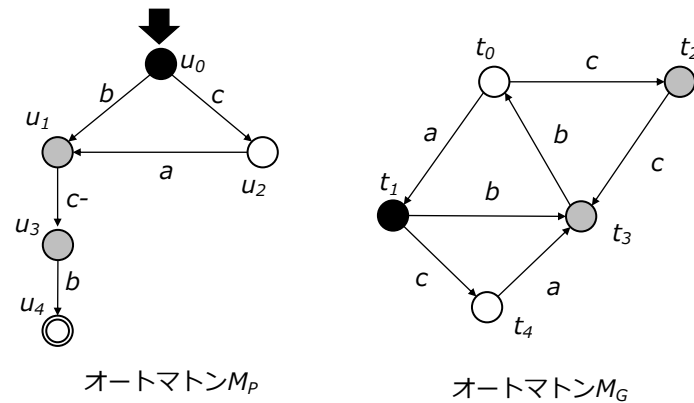


図 3.1.7: マッチング 3-1

同様にして、 t_2, t_3, t_4 を開始状態として処理を実行する。ここでは省略するが、いずれの場合も条件を満たす遷移ルートはなく、 M_G のどのノードを開始状態に見立てた場合においても条件を満たさないため充足不能と出力し終了する。

3.2 CPP の充足可能性判定

本節では, ShEx における CPP 問合せの充足可能性判定アルゴリズムに関して述べる.

以下, ShEx が *disjunction-capsuled* であるとする. *disjunction-capsuled* の定義を以下に示す.

- r, r_1, r_2, \dots, r_k を RBE とするとき, r 中の選言で接続された全ての RBE $r_1|r_2|\dots|r_k$ が '+' または '*' で修飾されている時, r は *disjunction-capsuled* であるという.
- ShEx $S = (\Sigma, \Gamma, \delta)$ において, 全ての $t \in \Gamma$ について $\delta(t)$ が *disjunction-capsuled* である時, S は *disjunction-capsuled* であるという.

ShEx が *disjunction-capsuled* でない場合, CPP の充足可能性問題は NP 困難であることが文献 [11] より示せる.

提案アルゴリズムを Algorithm 3.2.1 に示す.

Algorithm 3.2.1 CPP SATISFIABILITY DECISION

Input: ShEx schema $S = (\Sigma, \Gamma, \delta)$, Conjunctive Property Path Q

Output: “satisfiable” or “unsatisfiable”

```

1:  $M_G := \text{ShExToAutomaton}(S)$ ;
2:  $G_Q := \text{CppToGraph}(Q)$ ;
3:  $PQ := \text{SetQueue}(G_Q)$ ;
4:  $check := \text{true}$ ;
5:  $update\_check := \text{true}$ ;
6: while  $PQ$  is nonempty &  $check$  &  $update\_check$  & !Searchcheck( $G_Q$ ) do
7:    $x \leftarrow \text{dequeue}(PQ)$ ;
8:    $Q(x) := (x, p_1, y_1), \dots, (x, p_n, y_n) \in E(G_Q)$ ;
9:   for each  $(x, p, y) \in Q(x)$  do
10:     $check := \text{PP SATISFIABILITY DECISION}(M_G, p)$ ;
11:    if  $check$  then
12:      UpdateCand( $cands(x)$ );
13:      UpdateCand( $cands(y)$ );
14:       $update\_check = \text{Update}(G_Q, cands, y) \& \text{Update}(G_Q, cands, x)$ ;
15:      if ! $update\_check$  then
16:        report “unsatisfiable”;
17:      end if
18:       $PQ \leftarrow y$ ;
19:    else
20:      report “unsatisfiable”;
21:    end if
22:  end for
23: end while
24: report “satisfiable”;

```

まず, 1~2 行目で ShEx で記述されたグラフスキーマと CPP で記述された問合せをオートマトン M_G, G_Q に変換する. ShExToAutomaton は 3.1 節の通り ShEx をオートマトンに

変換する関数であり, CppToGraph は CPP を有向グラフ $G_Q = (V, E)$ と $cands, heads$ に変換する関数である. それぞれを以下のように定義する.

- CPP $Q = \wedge_i(x_i, p_i, y_i)$ を有向グラフ $G_Q = (V, E)$ と $cands, heads$ に変換する. ここで,
- V : 各変数 x_i, y_i の集合
- E : (x_i, p_i, y_i) の集合
- $cands$: 各ノードの候補を記憶するハッシュ
- $heads = \{v_1 | v_1 \in V, \text{いずれの } e \in E \text{ も } v_1 \text{ が入力ノードでない}\}$

次に, $PQ, check, update_check$ を以下のように定義する.

- PQ : 次に調べる変数を格納するキューであり, SetQue 関数は $heads$ がある場合には $heads$ を, ない場合には $V(G_Q)$ を返す.
- $check$: 現在判定している Property Path が充足可能であるかの真偽を格納する変数
- $update_check$: Update 関数から返る真偽を格納する変数

6 行目以降で各 Property Path の充足可能性判定を順に行う. Searchcheck 関数は G_Q の全ての Property Path が判定済みであれば true を, 未判定の Property Path があれば false を返す関数である.

PQ から取り出した変数 x が出力ノードとなる G_Q 上のエッジ集合を $Q(x)$ とする. $Q(x)$ の各 Property Path p を 3.1 節のアルゴリズムで判定する. ただし, ここでは PP SATISFIABILITY DECISION の充足可能か否かの出力を真偽値として $check$ を更新し, さらに充足可能な場合は $cands$ に候補を入れる. p が充足可能であれば, $cands(x), cands(y)$ の更新を行う. UpdateCand ではそれまでに格納していた候補と p の解として得られた候補との共通集合を取り, ノード候補を更新する. 14 行目の Update 関数は, 指定したノードを起点としてそのノードが入力ノードまたは出力ノードとして接続しているエッジを再帰的に辿り, 各ノードの候補を更新する関数である. 更新が完了した場合に true を, いずれかのノード候補が空となった場合は false を返し, その結果を $update_check$ に反映する. 更新の終了後, y を PQ に格納する.

p が充足不能である, もしくはいずれかの変数の候補がなくなると充足不能と出力する. 全ての Property Path の判定後に解となる組が存在すれば充足可能と出力する.

3.3 アルゴリズムの正当性

本節ではアルゴリズムの正当性を示す。

定理 I: Algorithm 3.2.1 の出力が「充足可能」であることと, CPP 問合せ Q が ShEx スキーマ S の下で充足可能であることは同値である。

証明の概要: Q のノードを u_1, u_2, \dots, u_n とする。

(\Rightarrow) Algorithm 3.2.1 が「充足可能」を出力し, その時点で $cands$ の各 u_i の候補として対応した型 t_i が格納されていたとする。 v_i を型 t_i をもつノードとすると, v_1, v_2, \dots, v_n を含む S に妥当なグラフで, v_1, v_2, \dots, v_n と各 Property Path を満たす v_i, v_j 間のノードから構成される部分グラフが Q を満たすものを構成できる。すなわち, Algorithm 3.2.1 の出力が「充足可能」であるならば, CPP 問合せ Q が ShEx スキーマ S の下で充足可能である。

(\Leftarrow) Q が S の下で充足可能であるとする。このとき, S に妥当なグラフ G で, Q を満たす部分グラフを含むものが存在する。この部分グラフを $G(Q)$ とする。 $G(Q)$ のノードのうち, Q の変数に対応しているノードの集合を $\{v_1, v_2, \dots, v_n\}$ とし, Q のノード u_i と $G(Q)$ のノード v_i が対応しているとする。

まず, s-typing の場合を考える。このとき, G において v_i は 1 つの型 (t_i と表す) をもつ。 $G(Q)$ と同型で, v_i を t_i に置き換えたグラフを G_t とし, 列 $(u_1, t_1), (u_2, t_2), \dots, (u_n, t_n)$ を考える。 Algorithm 3.2.1 ではいずれかの Property Path の解もしくは Q のノード候補が空となった時点で unsatisfiable を出力するが, $G(Q)$ が存在する場合にはいずれの Property Path も充足可能かつ $cands(u_n)$ には必ず t_n が格納されているため, while 内で unsatisfiable が出力されることなくループを抜ける。

次に, m-typing の場合を考える。このとき, v_i は複数の型をもつ可能性がある。 G における v_i の型集合を $\{t_{i1}, t_{i2}, \dots, t_{ik}\}$ とし, $G(Q)$ において v_i を $t_{i1i2\dots ik}$ で置き換えたグラフを G_t とする。 G は妥当なので, m-typing の定義から v_i は $t_{i1}, t_{i2}, \dots, t_{ik}$ をいずれも満たしており, $t_{i1i2\dots ik}$ の内容モデルは $t_{i1}, t_{i2}, \dots, t_{ik}$ の内容モデルの共通部分から構成される。 $G(Q)$ が存在する場合にはいずれの Property Path も充足可能かつ $cands(u_n)$ には必ず $t_{i1i2\dots ik}$ が格納されているため, while 内で unsatisfiable が出力されることなくループを抜ける。すなわち, s-typing か m-typing かを問わず, CPP 問合せ Q が ShEx スキーマ S の下で充足可能であると, Algorithm 3.2.1 の出力が「充足可能」である。

以上のことから両方向の同値性がいえるため, 定理 I は示される。 \square

第4章 評価実験

本章では、前章の提案手法の評価実験について述べる。

4.1 概要

RDF データに対して充足不能な問合せの実行を防ぐためには問合せ実行前に効率よく検出することが求められる。したがって、提案手法の実行時間と RDF データに対する問合せ時間を比較し、前者が十分に短いことを確認する必要がある。

まず、前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、SP²Bench, BSBM というベンチマークツールから作成したグラフデータ、及びそれに妥当な ShEx を用意した。そして、グラフデータと ShEx に対して充足不能な Property Path を作成し、両者の問合せに要した時間を計測した。Property Path の検索をグラフデータに対して行う際は、Ruby の sparql ライブラリを使用した。sparql ライブラリは Ruby 上で SPARQL 1.1 の検索を行えるようにするライブラリである。

アルゴリズムは Ruby 3.0.3 で実装し、全ての評価実験は Intel Core i5 CPU 2.3 GHz デュアルコア、16.00GB RAM、macOS Monterey 12.0.1 を搭載したマシンを用いて実行した。

SP²Bench [12] SP²Bench は、コンピュータ科学に関する書誌学ウェブサイトの DBLP に基づき任意のトリプル数もしくはデータサイズの RDF データを生成する SPARQL ベンチマークソフトである。図 4.1.1 は SP²Bench のデータ構造、図 4.1.2 は DBLP と SP²Bench のラベルの対応関係を表した図である (いずれも文献 [12] から引用している)。生成されるデータは、各行ごとに名前空間もしくはトリプルを表している。名前空間には 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 .) といったように要素を空白で区切って記述している。ノードは以下の 4 種類が存在する。

- 一般ノード：資源を識別するための IRI が表記されるノード
- 匿名ノード：「_ :」から始まるノード
- リテラルノード：「"」で囲まれており、文字や数値といった IRI でない値が表記されるノード
- クラスノード：先頭に名前空間の接頭辞「bench:」をつけて表記されるノード

ただし、SP²Bench から生成されるグラフデータは Notation3 というフォーマットを使用しているが、sparql ライブラリではこのフォーマットが対応していないため、対応しているフォーマットの一つである N-Triple へ変換している。変換には RDF Translator[14] というツールを利用した。本研究の評価実験では、SP²Bench で生成したトリプル数 10000 のデータを N-Triple に変換したデータを使用する。変換後のデータはトリプル数 10303、サイズ

が 1,715,705B である.

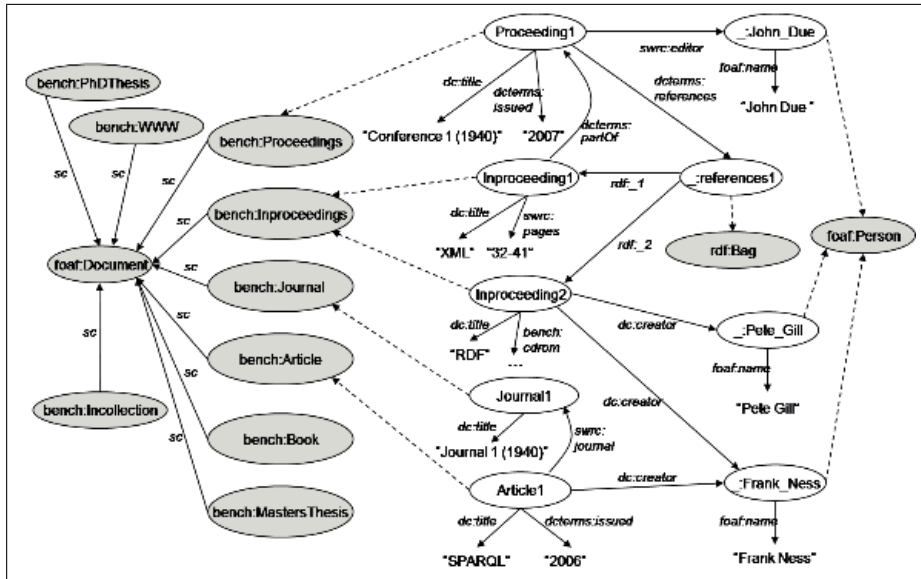


図 4.1.1: SP²Bench のデータ構造

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 4.1.2: DBLP と SP²Bench のラベルの対応関係

前述のグラフとラベルの構造に基づいて、m-typing の ShEx スキーマを以下のように定義する。ただし、接頭辞は省略している (例: <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>は type)。

$$\begin{aligned}
\Sigma &= \{\text{cdrom, abstract, booktitle, creator, publisher, title, issued, references, partOf,} \\
&\quad \text{homepage, name, seeAlso, type, rdf : _ , month, pages, note, editor, isbn, journal,} \\
&\quad \text{series, number, volume, chapter, address}\} \\
\Gamma &= \{l_1, l_2, l_3, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}\} \\
\delta(l_1) &= \varepsilon \\
\delta(l_2) &= \varepsilon \\
\delta(l_3) &= \varepsilon \\
\delta(t_1) &= \text{name} :: l_1 \\
\delta(t_2) &= \text{title} :: l_1 || \text{issued} :: l_2 || (\text{editor} :: l_3 | \text{editor} :: t_1)^* || \text{number} :: l_2^? || \text{volume} :: l_2^? \\
\delta(t_3) &= \text{abstract} :: l_1^? || \text{booktitle} :: l_1^? || \text{cdrom} :: l_1^? || \text{publisher} :: l_1^* || \text{title} :: l_1 || \text{references} :: t_4^* || \\
&\quad (\text{creator} :: l_3 | \text{creator} :: t_1)^* || \text{homepage} :: l_1^? || \text{seeAlso} :: l_1^* || \text{journal} :: t_2 || \text{month} :: l_2^? || \\
&\quad \text{note} :: l_1^? || \text{pages} :: l_2^? \\
\delta(t_4) &= (\text{rdf} : _ :: l_3 | \text{rdf} : _ :: t_3 | \text{rdf} : _ :: t_7)^+ \\
\delta(t_5) &= \text{booktitle} :: l_1 || \text{cdrom} :: l_1^? || \text{creator} :: t_1^+ || \text{publisher} :: l_1^* || \text{title} :: l_1 || \text{issued} :: l_2 || \\
&\quad \text{references} :: t_4^* || \text{homepage} :: l_1^? || \text{seeAlso} :: l_1^* || \text{chapter} :: l_2^? || \text{isbn} :: l_1^? || \text{pages} :: l_2^? \\
\delta(t_6) &= \text{booktitle} :: l_1^? || \text{publisher} :: l_1^* || \text{title} :: l_1 || \text{issued} :: l_2 || \text{homepage} :: l_1^? || \text{seeAlso} :: l_1^* || \\
&\quad \text{address} :: l_1^? || (\text{editor} :: l_3 | \text{editor} :: t_1)^+ || \text{isbn} :: l_1^? || \text{month} :: l_2^? || \text{note} :: l_1^? || \\
&\quad \text{number} :: l_2^? || \text{series} :: l_2^? || \text{volume} :: l_2^? \\
\delta(t_7) &= \text{abstract} :: l_1^? || \text{booktitle} :: l_1 || \text{cdrom} :: l_1^? || (\text{creator} :: l_3 | \text{creator} :: t_1)^+ || \\
&\quad \text{references} :: t_4^* || \text{title} :: l_1 || \text{issued} :: l_2 || \text{partOf} :: t_6^? || \text{homepage} :: l_1 || \text{seeAlso} :: l_1^* || \\
&\quad \text{number} :: l_2^? || \text{pages} :: l_2^? \\
\delta(t_8) &= \text{creator} :: t_1^+ || \text{title} :: l_1 || \text{homepage} :: l_1^? || \text{note} :: l_1^? \\
\delta(t_9) &= \text{creator} :: t_1^+ || \text{publisher} :: l_1 || \text{title} :: l_1 || \text{issued} :: l_2 || \text{homepage} :: l_1^? \\
\delta(t_{10}) &= \text{creator} :: t_1^+ || \text{publisher} :: l_1^+ || \text{title} :: l_1 || \text{issued} :: l_2 || \text{homepage} :: l_1^? || \text{seeAlso} :: l_1^* || \\
&\quad \text{isbn} :: l_1^? || \text{month} :: l_2^? || \text{number} :: l_2^? || \text{series} :: l_2^? || \text{volume} :: l_2^? \\
\delta(t_{11}) &= \text{booktitle} :: l_1^? || \text{cdrom} :: l_1^? || \text{creator} :: t_1^+ || \text{publisher} :: l_1^? || \text{title} :: l_1 || \text{issued} :: l_2 || \\
&\quad \text{references} :: t_4^* || \text{homepage} :: l_1^? || \text{seeAlso} :: l_1^* || \text{isbn} :: l_1^? || (\text{editor} :: l_3 | \text{editor} :: t_1)^+ || \\
&\quad \text{month} :: l_2^? || \text{series} :: l_2^? || \text{volume} :: l_2^?
\end{aligned}$$

元の型は以上の 14 つだが m-typing による新しい型の組合せを考慮する。 $\delta(t_2)$ と $\delta(t_6)$, $\delta(t_5)$ と $\delta(t_7)$, $\delta(t_9)$ と $\delta(t_{10})$ がそれぞれ組合せ可能なため、新しい型 $\delta(t_{26})$, $\delta(t_{57})$, $\delta(t_{910})$ を作成できる。

$$\begin{aligned}
\delta(t_{26}) &= \text{title} :: l_1 || \text{issued} :: l_2 || (\text{editor} :: l_3 | \text{editor} :: t_1)^+ || \text{number} :: l_2^? || \text{volume} :: l_2^? \\
\delta(t_{57}) &= \text{booktitle} :: l_1 || \text{cdrom} :: l_1^? || \text{creator} :: t_1^+ || \text{homepage} :: l_1 || \text{issued} :: l_2 || \text{pages} :: l_2^? || \\
&\quad \text{references} :: t_4^* || \text{seeAlso} :: l_1^* || \text{title} :: l_1 \\
\delta(t_{910}) &= \text{creator} :: t_1^+ || \text{publisher} :: l_1 || \text{title} :: l_1 || \text{issued} :: l_2 || \text{homepage} :: l_1^?
\end{aligned}$$

元の型と新しい型を合わせた 17 つの型を使用して実験を行う。

BSBM [13] BSBM は e コマースのユースケースに基づいた SPARQL の包括的なベンチマークツールである。図 4.1.3 は BSBM のデータ構造を表した図である ([13] から引用している)。実験ではサイズが 10,216,303 バイト (40,377 トリプル) の RDF データを生成した。生成されるデータは空白で区切られたトリプルで出力される。こちらも ShEx スキーマの定義が存在していなかったため作成した。semantics は s-typing である。SP²Bench のスキーマと同様に接頭辞は省略して記載している。

$$\begin{aligned}
\delta(t_t) &= \varepsilon \\
\delta(t_0) &= \varepsilon \\
\delta(t_1) &= \text{type} :: t_0 || \text{label} :: t_t || \text{comment} :: t_t | \text{subClassOf} :: t_1^? || \text{publisher} :: t_t || \text{date} :: t_t \\
\delta(t_2) &= \text{type} :: t_0 || \text{label} :: t_t || \text{comment} :: t_t | \text{publisher} :: t_t || \text{date} :: t_t \\
\delta(t_3) &= \text{type} :: t_0 || \text{label} :: t_t || \text{comment} :: t_t | \text{homepage} :: t_t || \text{country} :: t_t | \text{publisher} :: t_3 || \text{date} :: t_t \\
\delta(t_4) &= (\text{type} :: t_0 | \text{type} :: t_1) + || \text{label} :: t_t || \text{comment} :: t_t | \text{producer} :: t_3 || \\
&\quad \text{productPropertyNumeric1} :: t_t || \text{productPropertyNumeric2} :: t_t || \\
&\quad \text{productPropertyNumeric3} :: t_t || \text{productPropertyNumeric4} :: t_t^? || \\
&\quad \text{productPropertyNumeric5} :: t_t^? || \text{productPropertyTextual1} :: t_t || \\
&\quad \text{productPropertyTextual2} :: t_t || \text{productPropertyTextual3} :: t_t || \\
&\quad \text{productPropertyTextual4} :: t_t^? || \text{productPropertyTextual5} :: t_t^? || \\
&\quad \text{productFeature} :: t_2^* || \text{publisher} :: t_3 || \text{date} :: t_t \\
\delta(t_5) &= \text{type} :: t_0 || \text{label} :: t_t || \text{comment} :: t_t | \text{homepage} :: t_t || \text{country} :: t_t | \text{publisher} :: t_5 || \text{date} :: t_t \\
\delta(t_6) &= \text{type} :: t_0 || \text{product} :: t_4 | \text{vendor} :: t_5 | \text{price} :: t_t || \text{validFrom} :: t_t | \text{validTo} :: t_t || \\
&\quad \text{deliveryDays} :: t_t | \text{offerWebpage} :: t_6 | \text{publisher} :: t_5 || \text{date} :: t_t \\
\delta(t_7) &= \text{type} :: t_0 | \text{name} :: t_t | \text{mbox_sha1sum} :: t_t || \text{country} :: t_t | \text{publisher} :: t_t || \text{date} :: t_t \\
\delta(t_8) &= \text{type} :: t_0 || \text{reviewFor} :: t_4 | \text{reviewer} :: t_7 | \text{reviewDate} :: t_t || \text{title} :: t_t | \text{text} :: t_t || \\
&\quad \text{rating1} :: t_t^? || \text{rating2} :: t_t^? || \text{rating3} :: t_t^? || \text{rating4} :: t_t^? || \text{publisher} :: t_t || \text{date} :: t_t
\end{aligned}$$

$\delta(t_t)$ 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。

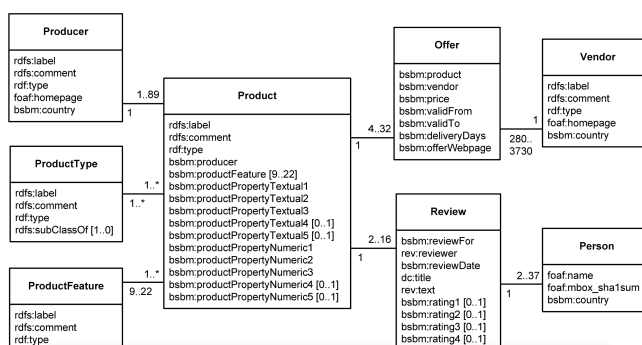


図 4.1.3: BSBM のデータ構造

実験に用いる CPP 問合せは Ruby で実装した自動生成プログラムによって作成した。プログラムは以下の流れで実行する。

1. 作成する CPP 問合せのエッジ数 ($edge_{num}$), Property Path あたりの長さ ($pplen_{num}$), オペレーター ($?, +, *$) をラベルに付与する確率 ($prob_{num}$) を指定する。
2. ラベルをランダムに $pplen_{num}$ 個取得し, $prob_{num}$ の確率でオペレーターを付与する。以上のラベルを連結して1つの Property Path とする。この操作を $edge_{num}$ 回繰り返す。
3. 2で生成された $edge_{num}$ 本の Property Path を連結する。ここでは指定した CPP の形状 (直鎖状, 環状等) に従って連結を行う。

上記のプログラムを用いてサイズ (=エッジ数) 毎に 10 個の問合せを作成した。本実験では $pplen_{num} = 5$, $prob_{num} = 0.05$ とし, CPP の形状については制限なくランダムに形成されるものとしている。

4.2 結果

ShEx スキーマに対して提案手法を, SP²Bench と BSBM から生成した RDF データに対しては sparql ライブラリで問合せを実行した。SP²Bench データセットの結果を表 4.1 に, BSBM データセットの結果を表 4.2 に示す。サイズ毎に作成した 10 個の CPP 問合せでの実行時間の平均をそのサイズの実行時間とする。計測には Ruby の Benchmark モジュールを用いており, 実行時間の単位は秒である。

表 4.1: 結果: SP²Bench dataset

問合せのサイズ (エッジ数)	5	10	15	20	25
(a) 提案手法	0.0182	0.0171	0.0186	0.0249	0.0222
(b) RDF データ問合せ	14.4	31.7	40.7	56.5	63.9
時間比率 (a/b)	0.00126	0.000539	0.000457	0.000441	0.000347

表 4.2: 結果: BSBM dataset

問合せのサイズ (エッジ数)	3	4	5	6
(a) 提案手法	0.0255	0.0268	0.0233	0.0281
(b) RDF データ問合せ	26.3	94.6	139	297
時間比率 (a/b)	0.000970	0.000283	0.000168	0.0000946

4.3 考察

実験結果より，RDF データに対する問合せ時間に対して，提案手法は非常に短い時間で実行できることが確認できた．RDF データでの問合せ時間は問合せのサイズが大きくなるとそれに伴って大きくなる傾向にあるが，提案手法ではそれほど大きな時間の増加は見られない．また，時間比率の値を確認すると，提案手法の実行時間はほとんど無視できるほど小さいことが分かる．

問合せの実行時間はデータのサイズに対して増えるため，大きなデータで実行した場合の時間比率はより小さくなると考えられる．実験で使用した生成データは 10000 トリプルと実際のものよりサイズが小さいデータである．実前にごく短い時間で充足可能性を判定できるため，充足不能な問合せを検出する本研究のアルゴリズムは充足不能な場合の時間を大幅に節約できると考えられる．また，充足可能な場合であっても充足可能性は迅速に判定できるため，提案手法を実行することで大きな時間の損失が発生することはない．

第5章 まとめ

本論文では ShEx の下での CPP 問合せ充足可能性判定アルゴリズムを提案した。評価実験の結果は、提案手法による充足可能性判定がグラフデータに対する問合せよりも短い時間で効率的に行っていたことを示している。ただし、実データを用いての実験については不十分な点があり、追加実験を行う必要がある。

また、ShEx の否定等の機能に触れておらず、型定義にも制限を加えていたため、それらへの対応も必要と考えられる。アルゴリズムの観点ではさらなる高速化や使用メモリの効率化といった改良点が挙げられる。

謝辞

本研究を進めるにあたり，終始適切なお助言を賜り，また暖かく見守ってくださった鈴木伸崇先生に深く感謝いたします。先生のお助言がなければ本研究・本論文は完成いたしませんでした。併せて共同研究室でお世話になり，沢山のお助言を示してくださった阪口哲男先生に深く感謝いたします。また，ともに研究に励んだ先輩方を始めとする鈴木伸崇研究室のメンバーの皆様にも，多くのご指摘や議論をいただきました。心より感謝を申し上げます。

参考文献

- [1] Dan Brickley, R.V. Guha. RDF Schema 1.1. <https://www.w3.org/TR/rdf-schema/>, (accessed 2021-12-10).
- [2] Eric Prud'hommeaux, Iovka Boneva, Jose Emilio Labra Gayo, Gregg Kellogg. Shape Expressions Language 2.next. <https://shexspec.github.io/spec/>, (accessed 2021-12-10).
- [3] Thornton, K., Solbrig, H., Stupp, G. S., Labra Gayo, J. E., Mietchen, D., Prud'hommeaux, E., and Waagmeester, A. Using shape expressions (ShEx) to share RDF data models and to guide curation with rigorous validation. In Hitzler, P., Fernandez, M., Janowicz, K., Zaveri, A., Gray, A. J., Lopez, V., Haller, A., and Hammar, K., editors, Proceedings of the European Semantic Web Conference, 2019. pp. 606–620.
- [4] Andy Seaborne. SPARQL 1.1 Property Paths. <https://www.w3.org/TR/sparql11-property-paths/>, (accessed 2021-12-10).
- [5] Benedikt, M., Fan, W., and Geerts, F. XPath satisfiability in the presence of DTDs. *J. ACM*, 2008, 55(2):8:1–8:79.
- [6] Figueira, D. Satisfiability of XPath on data trees. *ACM SIGLOG News*, 2018, 5(2):4–16.
- [7] Zhang, X., den Bussche, J. V., and Picalausa, F. On the satisfiability problem for SPARQL patterns. *Journal of Artificial Intelligence Research*, 2016, 55:403–428.
- [8] Matsuoka, S. and Suzuki, N. Detecting unsatisfiable pattern queries under shape expression schema, Proceedings of the 16th International Conference on Web Information Systems and Technologies, 2020, pp. 285–291.
- [9] Slawek Staworko, Iovka Boneva, José Emilio Labra Gayo, Samuel Hym, Eric G. Prud'hommeaux, Harold R. Solbrig. Complexity and Expressiveness of ShEx for RDF, Proc. ICDT, 2015, pp.195-211.
- [10] 近藤嘉雪. 定本 C プログラマのためのアルゴリズムとデータ構造. 第2版, ソフトバンク, 1992, 414p.
- [11] E. Kopczynski and A. To. Parikh images of grammars: Complexity and applications, In LICS, 2010, pp. 80-89.
- [12] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. SP2Bench: a SPARQL Performance Benchmark, Proc. ICDE, 2009, pp. 371-393.
- [13] Christian Bizer, Andreas Schultz. The Berlin SPARQL Benchmark. *International Journal on Semantic Web & Information Systems*, 2009, 5(2), pp. 1-24.

- [14] Alex Stolz. RDF Translator. <http://rdf-translator.appspot.com/>, (accessed 2021-12-10).