有向グラフに対する Shape Expression Schema の妥当性検証及び修正手法の提案

藤永　健† 鈴木　伸崇††

† 筑波大学大学院図書館情報メディア研究科　〒 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: †s1821627@s.tsukuba.ac.jp, ††nsuzuki@slis.tsukuba.ac.jp

あらまし　近年，RDF のスキーマとして Shape Expression Schema (ShEx) が提案されている．ShEx は型を Regular Bag Expression で定義する．グラフの各ノードに対して，ノードの構造が型の定義を満たしてる場合，その型を割り当てることができる．従来の RDF Schema と比較して，スキーマ言語としての形式的なセマンティクスが定義されているため，データ構造をより厳密に定義することができる．本研究では，グラフデータに対する ShEx の妥当性検証アルゴリズムを示す．また，グラフデータが妥当でないノードを含む場合，どのようにノードやエッジを修正すればよいかを提示できれば有用である．そこで，妥当でないノードに対して，そのノードに最適な型とその修正手法を提案する方法も示す．

キーワード　Shape Expression Schema, グラフ，データ構造，RDF，妥当性検証

1 はじめに

グラフデータの構造はスキーマによって表現できる．スキーマを定義することで，問い合わせ式の記述に役立てることができるなどの利点がある．しかし，グラフデータがスキーマに対して妥当でなければ，その恩恵を受けることはできない．妥当性検証を行い，グラフデータの妥当性を保証することで，スキーマを利用した処理を確実に行うことができる．

グラフデータに対するスキーマとして Shape Expression Schema (以下，ShEx) が提案されている [1]．既存の RDF Schema と比較して，ShEx は妥当性に関して形式的なセマンティクスが定義されているため，より厳密にデータ構造の定義と妥当性検証を行うことができる．ShEx は型の集合として記述されるスキーマであり，グラフデータの各ノードに対してその型の定義を満たしている型を割り当てることができるとき，そのグラフデータは ShEx に妥当となる．

本稿は，グラフデータに対する ShEx の妥当性検証アルゴリズムおよび修正手法を提案する．提案アルゴリズムは，サイズの大きいグラフデータも検証できるように，グラフデータ全体をメモリ上に保持しない点に特徴がある．また，効率性の向上及びメモリの消費量を更に抑えるため，ShEx の型の依存関係に着目し，階層の概念を導入した妥当性検証を行うアルゴリズムも提案する．さらに，修正手法は，グラフデータが妥当でない場合，妥当でないノードに最適な型とその修正方法を提示する．これは，文字列に対する正規表現の修正方法をグラフと型に適用させたものである．

評価実験の結果，提案アルゴリズムの実行時間がほぼ線形であり，グラフデータに対してメモリ消費量が比較的小さく抑えられているという結果が得られた．また階層化を用いたアルゴリズムの方がやや省メモリであることも確認できた．また，妥当でないノードの修正に関しては，誤りが少数であれば，概ね所望の修正が可能であるという結果が得られた．

関連研究

ShEx の妥当性検証アルゴリズムは文献 [2], [3] で考察されているが，妥当性検証が理論上可能であることを示すためのものであり，効率については特に考慮されていない．本稿は，ノードへ割り当てる型集合をより限定されたものにすることなどにより，より効率的なアルゴリズムを目指している．また，妥当でないデータの修正手法の提案については，RDF データの構文誤りを修正する研究 [4] が存在しているが，スキーマレベルの修正ではない．また ShEx 以外のスキーマでは，XML のスキーマである DTD を参照して妥当でない XPath 式を修正する研究 [5] も存在する．著者の知る限り，ShEx を対象としたスキーマレベルでのグラフデータを修正するアルゴリズムはまだ提案されていない．

2 諸定義

ラベルの集合を Σ とする．Σ 上のグラフ (以下，単にグラフ) を 2 次組 $G = (V, E)$ と表す．ここで，V はノードの集合，$E \subseteq V \times \Sigma \times V$ はラベル付き有向辺 (以下エッジ) の集合である．簡単な例として，図 1 にグラフ G_1 を示す．ここで，

$$\Sigma = \{a, b, c\},$$
$$V_1 = \{n_0, n_1, n_2, n_3, n_4\},$$
$$E_1 = \{(n_0\, a\, n_1), (n_0\, b\, n_2), (n_0\, a\, n_3), (n_1\, b\, n_3), (n_2\, c\, n_4)\}$$

である．

エッジを出力しているノードを出力ノード，そのエッジに付いているラベルを出力ラベル，エッジの向かう先である受け取

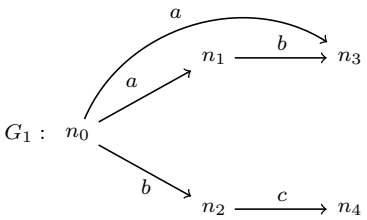


図 1　ラベル付き有向グラフの例

り側のノードを入力ノードと呼称する．グラフの各ノードは出力ノードでもあり，入力ノードでもあり得る．ここで，グラフ $G = (V, E)$ におけるノード n から出ているエッジの集合，すなわち出力ラベルとその入力ノードの集合を，

$$\mathit{out\text{-}lab\text{-}node}_G(n) = \{(a, m) \in \Sigma \times V \mid (n, a, m) \in E\}$$

と定義する．例えば，図 1 では

$$\mathit{out\text{-}lab\text{-}node}_{G_1}(n_0) = \{(a, n_1), (b, n_2), (a, n_3)\}$$

である．ただし，ノードによっては同じ出力ラベルや近傍入力ノードから成るエッジを複数持つ可能性もある．その場合，重複を許さない集合ではグラフを定義することはできない．同様に，ノード n の出力ラベルに関しても，集合では各ラベルの出現回数を示すことは不可能である．実際 G_0 における，n_0 の出力ラベルの集合は $\{a, b\}$ であるが，ラベル a が 2 回出現していることまでは示されていない．このように集合では表現に限界があるため，シンボルの出現回数を規定する bag を導入する．

シンボルの集合を Δ とする．Δ に対する bag とは，シンボルをその出現回数に対応させる関数 $w : \Delta \to \mathbb{N}$ のことである．集合が $\{a, \cdots\}$ で表現されるのに対して，bag は $\{|a, \cdots|\}$ で表現される．例えば，n_0 の出力ラベルは集合では $\{a, b\}$ となるが，bag では $\{|a, a, b|\}$ と表現できる．この bag をグラフに適用すると，グラフ $G = (V, E)$ におけるノード n についての出力ラベルは，

$$\mathit{out\text{-}lab}_G(n) = \{|a \mid (n, a, m) \in E|\}$$

と定義できる．

Regular Bag Expression と ShEx

XML のスキーマである DTD や XML Schema など，多くのスキーマでは Regular Expression を用いて内容モデルを定義している．一方，グラフではノード間の順序を考慮しないため，bag に基づく言語を表現できる式が望ましい．そこで，Regular Bag Expression (以下，RBE) という概念を導入する [2]．RBE は，論理和 | と順序を無視した連結 $\|$ と $*$ を用いて bag を定義する．Δ に対する RBE は以下の文法 E で定義される．

$$E ::= \epsilon | a | E^* | (E\text{"}|\text{"}E) \quad (a \in \Delta)$$

また，$E^? = \epsilon | E$，$E^+ = e \parallel E^*$ である．

RBE E の言語は $L(E)$ と表され，次のように定義される．

$$L(E_1|E_2) = L(E_1) \cup L(E_2)$$
$$L(E_1 \parallel E_2) = (E_1) \uplus L(E_2)$$
$$L(E^*) = \bigcup_{i \geqq 0} L(E)^i$$
$$L(E^{[n,m]}) = \bigcup_{n \leqq i \leqq m} E^i$$

ShEx は，3 次組 $S = (\Sigma, \Gamma, \delta)$ であり，ここで Σ はラベルの集合，Γ は型の集合，δ は型を定義する関数であり，次を満たす．

$$\delta : \Gamma \to \Sigma \times \Gamma \text{上の bag の集合 (RBE)}$$

以下に ShEx の例を示す．ただし，$(a, t) \in (\Sigma, \delta)$ を以下 $a :: t$ と表記する．

例 1. ShEx $S_1 = (\Sigma, \Gamma, \delta)$

$$\begin{aligned}
\Sigma = &\ \{\text{a, b, c}\},\\
\Gamma = &\ \{t_0, t_1, t_2, t_3\},\\
\delta(t_0) \to &\ \epsilon,\\
\delta(t_1) \to &\ (a :: t_1 | a :: t_2)^+ \parallel b :: t_3,\\
\delta(t_2) \to &\ b :: t_0,\\
\delta(t_1) \to &\ c :: t_0
\end{aligned}$$

グラフデータの妥当性

グラフが ShEx に妥当であるとき，グラフの全てのノードに型の定義を満たすような型を割り当てることができる．このとき，ノードに割り当てられる型は 1 つだけという制限はなく，複数でも許されている．ここで，全てのノードに型を 1 つだけ割り当てる場合は single-type，2 つ以上割り当てることを許す場合は multi-type と呼称する．

以下，ShEx $S = (\Sigma, \Gamma, \delta)$ の下でのグラフ $G = (V, E)$ の妥当性の定義を示す．そのために，いくつかの関数及び概念を導入する．まず，ノードに型を割り当てる関数を λ と表す．ここで，

$$\text{single-type} : V \to \Gamma$$
$$\text{multi-type} : V \to 2^\Gamma$$

である．

次に，出力ノードと入力ノードの集合 $\mathit{out\text{-}lab\text{-}node}_G(n)$ について，λ を用いて拡張し，出力ラベルと入力ノードの型の bag，

$$\mathit{out\text{-}lab\text{-}type}^\lambda_G(n) = \{|a :: \lambda(m) \mid (n, a, m) \in E|\}$$

と定義する．ただし，multi-type の場合は $\lambda(m)$ の要素を複数もつ可能性があるため，$\mathit{out\text{-}lab\text{-}type}^\lambda_G(n)$ を平坦化した $\mathit{fl\text{-}out\text{-}lab\text{-}type}^\lambda_G(n)$ を導入する．そのために，bag $w(\Sigma \times 2^\Gamma)$ の平坦化関数 $\mathit{Flatten}$ を，

$$\mathit{Flatten}(w) = \|_{a::T \in\!\!\!\in w} \ (|_{t \in T}\ a :: t)$$

と定義する．ここで，$a :: T \in\!\!\!\in w$ はシンボル $a :: T$ が $w(a :: T)$

回出現することを意味している．例えば，bag$\{|a :: \{t_0, t_1\}, b :: t_2|\}$ では，

$$Flatten(\{|a :: \{t_0, t_1\}, b :: t_2|\}) = (a :: t_0 | a :: t_1) \parallel (b :: t_2)$$

である．この関数 $Flatten$ を用いることで，

$$\textit{fl-out-lab-type}^{\lambda}_G(n) = L(Flatten(\textit{out-lab-type}^{\lambda}_G(n)))$$

と定義することができる．ここまでで定義した概念を用いて，λ が妥当な型付けであるための条件を以下に示す．

single-type

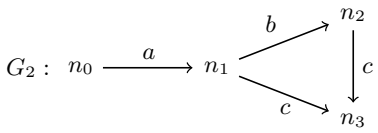
$\forall n \in V \mid \lambda(n) \neq nil \wedge \textit{out-lab-type}^{\lambda}_G(n) \in \delta(\lambda(n))$

multi-type

$\forall n \in V, \forall t \in \lambda(n) \mid \lambda(n) \neq \emptyset \wedge \textit{fl-out-lab-type}^{\lambda}_G(n) \cap \delta(t) \neq \emptyset$

multi-type の例を以下に示す．ただし，ShEx は Σ, Γ を省略する．

例 2. multi-type



$$\begin{aligned} S_2 : \delta(t_0) &\to a :: t_1 \\ \delta(t_1) &\to b :: t_2 \parallel c :: t_3 \\ \delta(t_2) &\to c :: t_3{}^{?} \\ \delta(t_3) &\to \epsilon \end{aligned}$$

ここで，次のように定義される λ を考える．

$$\lambda(n_0) = \{t_0\},\ \lambda(n_1) = \{t_1\},\ \lambda(n_2) = \{t_2\},\ \lambda(n_3) = \{t_2, t_3\}$$

このとき，n_1 について，

$$\textit{fl-out-lab-type}^{\lambda}_{G_2}(n_1) = \{|b :: t_2, c :: t_2|, |b :: t_2, c :: t_3|\}$$

となり，

$$\textit{fl-out-lab-type}^{\lambda}_{G_2}(n_1) \cap \delta(t_1) = \{|b :: t_2, c :: t_3|\} \neq \emptyset$$

であるため，ノード n_1 は型 t_1 に妥当である．同様に他のノードについても考えていくと，上記の条件を満たすため，グラフ G_2 は S_2 に妥当であるといえる．

3 妥当性検証

本節では，妥当性検証の提案アルゴリズムについて述べる．single-type の場合，型として非常に限定された RBE を用いても計算困難である [2]．そのため，提案アルゴリズムは multi-type を想定している．

出力ノード	出力ラベル	入力ノード
$n1$	a	$n2$
$n1$	a	$n3$
$n1$	b	$n4$
$n2$	c	$n5$
$n2$	d	$n6$
$n3$	c	$n5$
$n3$	d	$n7$
$n4$	d	$n6$
$n4$	d	$n8$

図 2　入力グラフデータの例

入力グラフデータ

本稿で扱うグラフデータは，RDF データの形式である N-triples 形式に準じた形でファイルに格納されていることを前提とする．すなわち，グラフデータの各行はエッジであり，出力ノード，出力ラベル，入力ノードの順で構成されている．図 2 にグラフデータの例を示す．

妥当性検証を行っていく際は，グラフデータが大規模である場合を考慮し，グラフデータ全体をメモリ上に置くのではなく，出力ノード毎に出力エッジを読み込み検証を行っていく．例では，最初に読み込むのは出力ノードが $n1$ である 1〜3 行目，次に読み込むのは 4〜5 行目である．ただし，グラフデータによっては同一出力ノード毎にエッジが並んでいないため，あらかじめ外部ソートで並び替えを行っておく．

λ の初期設定

まず，入力グラフデータに対して λ の初期定義を行う．既存手法 [2], [3] では λ の初期値として各ノードに ShEx の全ての型を割り当てているが，これでは $\textit{fl-out-lab-type}^{\lambda}_G(n)$ のサイズが膨大になり，妥当性検証の時間およびメモリ消費量も大きく増加してしまう．そこで，ノードの出力ラベル集合 $\textit{out-lab}_G(n)$ と型のラベル集合 $\Sigma(t)$ に注目する．ここで，$\Sigma(t)$ は t に出現するラベルの集合である．RBE によっては出力ラベルが 0 回の出現もあり得るため，$\textit{out-lab}_G(n)$ が $\Sigma(t)$ の部分集合である場合，$\lambda(n)$ に型を追加する．しかし，妥当でないグラフデータの場合 $\lambda(n)$ が妥当性検証前に空となる可能性がある．その場合，妥当性検証の際に判別しやすくするため，エラーであることを示す型 t_{error} を $\lambda(n)$ に付与する．また，出力エッジをもたないノードは，$\delta(t) = \epsilon$ である型を付与し，リテラルノードには対応するリテラル型を付与する．ただし，これらのノードは簡単に区別できるため，$\lambda(n)$ としてノード毎には定義しない．これらのことを踏まえて，λ の初期定義を行うアルゴリズム initialization を示す (Algorithm 1)．

妥当性検証アルゴリズム

以上で，グラフデータを格納するファイル (以下，グラフファイル) と λ の初期定義が得られた．次に，これらと ShEx を用いて妥当性検証を行うアルゴリズムを示す．まず，以下に妥当性検証アルゴリズム validation を示す (Algorithm 2)．このアルゴリズムでは，各ノード $n \in V$ に対して，9 行目の isvalid(n, t)

Algorithm 1 initialization

Input: グラフデータ $G = (V, E)$, ShEx $S = (\Sigma, \Gamma, \delta)$
Output: 型の割り当て λ

```
1: λ ← ∅
2: for each n ∈ V do
3:     for each t ∈ Γ do
4:         if out-lab_G(n) ⊆ Σ(t) then
5:             λ(n) ← λ(n) ∪ {t}
6:         end if
7:     end for
8:     if λ(n) = ∅ then
9:         λ(n) ← {t_error}
10:    end if
11: end for
```

が成り立たない，すなわち，その時点で n が t に関して妥当でない場合，t を $\lambda(n)$ から除く，という処理を λ が変化しなくなるまで繰り返す．

Algorithm 2 validation

Input: グラフファイル ($G = (V, E)$), ShEx $S = (\Sigma, \Gamma, \delta)$, 型の初期割り当て λ
Output: true or false

```
1: λ_old ← ∅
2: while λ ≠ λ_old do
3:     λ_old ← λ
4:     while グラフファイルが EOF に到達していない do
5:         出力ノードが同一である行を読み込む．そのノードを n とする．out-lab-node_G(n) を取得する
6:         λ を参照し，fl-out-lab-type^λ_G(n) を取得する．
7:         F ← ∅
8:         for each t ∈ λ(n) do
9:             if isvalid(n, t) then
10:                F ← F ∪ {t}
11:            end if
12:        end for
13:        if F = ∅ then
14:            return false
15:        end if
16:        λ(n) ← λ(n) ∩ F
17:    end while
18: end while
19: return true
```

このアルゴリズムにおいて，9 行目の isvalid は

$$\mathit{fl\text{-}out\text{-}lab\text{-}type}^{\lambda}_{G}(n) \cap \delta(t) \neq \emptyset \qquad (1)$$

であるか否かを判定している (Algorithm 3 として後述)．なお，$\delta(t)$ が一般の RBE である場合，この判定を効率良く行うことは困難である [2]．そのため本稿では，$\delta(t)$ が以下の形に限定されていると仮定する．

$$\delta(t) = E_1 \parallel E_2 \parallel ... \parallel E_l$$

ここで，

$$E_i = (a_{i_1} :: t_{i_1} \mid a_{i_2} :: t_{i_2} \mid ... \mid a_{i_k} :: t_{i_k})^{[n_i, m_i]}$$

である ($1 \leqq i \leqq l$)．ただし，この制限を設け，かつ $\mathit{fl\text{-}out\text{-}lab\text{-}type}^{\lambda}_{G}(n)$ のサイズを 1 に限定しても，式 (1) が成立するか否かは NP 困難であることが 3SAT からの帰着により示せる．しかし，実際のスキーマの型においては，同じ「ラベル::型」が複数回出現することは稀であると考えられる．そこで本稿では，任意の $t \in \Gamma$ に対して，$\delta(t)$ において同じ「ラベル::型」が複数回出現することはないと仮定する．

このとき，式 (1) は次のアルゴリズムで効率よく判定することができる (Algorithm 3)．1 行目の $\mathit{fl\text{-}out\text{-}lab\text{-}type}^{\lambda}_{G}(n)$ の要素は「ラベル::型」の bag g である．2 行目において，g に出現する「ラベル::型」の集合を $d(g)$ と表す．例えば，$d(|a :: t_1, a :: t_1, b :: t_2|) = \{a :: t_1, b :: t_2\}$ である．また，$d(g, E_i)$ は次のように定義される．

$$d(g, E_i) = \{a :: t \in d(g) \mid a :: t \text{ は } E_i \text{ に出現 }\}$$

また，$w_g(a :: t)$ は，g における $a :: t$ の出現回数を表す．以上より，2 行目の条件は，各 E_i に対して，g に出現する「ラベル::型」で E_i に適合するものの出現回数の合計が，E_i が指定する出現回数 $[n_i, m_i]$ を満たしているか否かを判定している．

Algorithm 3 isvalid

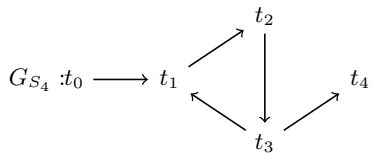
Input: ノード n, 型 t
Output: true or false

```
1: for each g ∈ fl-out-lab-type^λ_G(n) do
2:     if for every 1 ≦ i ≦ l, n_i ≦ Σ_{a::t∈d(g,E_i)} w_g(a :: t) ≦ m_i then
3:         return true
4:     end if
5: end for
6: return false
```

ShEx の階層化と妥当性検証アルゴリズム

妥当性検証アルゴリズム validation では，$\mathit{fl\text{-}out\text{-}lab\text{-}type}^{\lambda}_{G}(n)$ の要素が多いほど処理の時間が増加する．またその生成方法である，$\mathit{out\text{-}lab\text{-}type}^{\lambda}_{G}(n)$ の平坦化にかかる時間も当然大きく増加する．これは入力ノードの λ によって結び付けられる型が多いほど顕著に現れる．initialization アルゴリズムでは，型のラベル状態によって，ほぼ全ての型が $\lambda(n)$ に付与されるおそれもあるが，実際のグラフファイルでは，このノードがそのまま入力ノードで現れる可能性が十分にある．そこで，型同士の依存関係に着目し，ShEx の階層化を行う．先に入力ノード型が確定していれば，その型に依存している，すなわちエッジをもつ型が付与されているノードの検証時，$\mathit{fl\text{-}out\text{-}lab\text{-}type}^{\lambda}_{G}(n)$ の要素を最小に抑えることができる．階層を求めるため，最初に ShEx からスキーマグラフ G_S を抽出する．各型において，リテラル型以外の定義部分からグラフを作成する．例えば，次の ShExS_4 から抽出されるグラフは以下の通りである．

$$
\begin{aligned}
S_4 : \delta(t_0) &\to a :: t_1 \\
\delta(t_1) &\to b :: t_2 \\
\delta(t_2) &\to c :: t_3 \\
\delta(t_3) &\to a :: t_1 \parallel d :: t_4 \\
\delta(t_4) &\to \epsilon
\end{aligned}
$$



このグラフから階層，すなわち依存関係を得るために強連結成分分解を行う．この例では，以下の 3 階層を得ることができる．ここで，$\{t_4\}$ が最も下の階層であり，$\{t_0\}$ が最も上の階層である．

$$R = [\{t_4\}, \{t_1, t_2, t_3\}, \{t_0\}]$$

そして，下の階層から上にむけて階層ごとに妥当性検証を行っていく．具体的には，$\lambda(n)$ に該当する階層の要素を含んでいる場合に限り isvalid を行うことで，入力ノードの型がほぼ確定された状態で検証することができる．次の階層に移った際，再度ファイルを先頭から読み込んでいく．これを全ての階層について行う．

階層化妥当性検証アルゴリズム hierarchical validation を示す (Algorithm 4)．Algorithm 2 同様，入力の λ は initialization で得られたものである．3 行目で r は下の階層から順に R から取り出され，r に関する妥当性検証を 4〜21 行目で行っている．7〜9 行目では，$\lambda(n)$ が階層 r から外れていた場合，n の検証をスキップしている．他は Algorithm 2 と同様である．

4 妥当でないノードの修正手法

前節の妥当性検証において，妥当でないノードが含まれた場合，そこで妥当性検証が終了する．その際，妥当でない出力ノードのエッジを修正するなどした後，再度はじめから妥当性検証をしなければならない．そこで，妥当性検証が途中で終了した場合，その出力ノードに対して，スキーマ構造的に最適だと思われる型を推定し，その型に妥当となるような修正方法を導く．ここでの修正とは，データ構造に関する修正であり，出力ラベルや入力ノードの型を別の型に変更することや，エッジの追加及び削除が該当する．このアルゴリズムによって推定された型を $\lambda(n)$ に付与し，妥当性検証を再開するという用途を想定している．妥当性検証が終了して，グラフデータが ShEx に妥当となった後，修正する出力ノードの *out-lab-node*$_G(n)$ 及び，推定した型とその修正方法を示す．以下では，簡単のため single-type に基づいて説明するが，multi-type にも容易に拡張可能である．

関連研究として，Regular Expression に対してマッチしない文字列を，マッチするように修正する方法を求める手法 [6] が存在する．この手法の入力と出力は以下の通りである．

Algorithm 4 hierarchical validation

Input: グラフファイル ($G = (V, E)$), ShEx $S = (\Sigma, \Gamma, \delta)$, 型の初期割り当て λ, 型の階層 R
Output: 妥当性判定 (true or false)，λ

```
1: λ_old ← ∅
2: while λ ≠ λ_old do
3:     for each r ∈ R do
4:         λ_old ← λ
5:         while グラフファイルが EOF に到達していない do
6:             出力ノードが同一である行を読み込む．そのノードを n とする．out-lab-node_G(n) を取得する
7:             if λ(n) ∩ {r} = ∅ then
8:                 next
9:             end if
10:            λ を参照し，fl-out-lab-type^λ_G(n) を取得する．
11:            F ← ∅
12:            for each t ∈ λ(n) do
13:                if isvalid(n, t) then
14:                    F ← F ∪ {t}
15:                end if
16:            end for
17:            if F = ∅ then
18:                return false
19:            end if
20:            λ(n) ← λ(n) ∩ F
21:        end while
22:    end for
23: end while
24: return true
```

Input: Regular Expression r, 文字列 s
Output: スタートからゴールまでのどの経路も s を r を満たすように修正した文字列を表し，経路の重みが修正コストと一致するような有向グラフ

本稿では，この手法を利用して妥当でなかった出力ノードの *out-lab-type*$^{\lambda}_G(n)$ を型 $\delta(t)$ に修正する方法を求める．この際，RBE である $\delta(t)$ を Regular Expression に，集合である *out-lab-type*$^{\lambda}_G(n)$ を文字列に対応付ける必要がある．この対応付けを行う方法を以下に示す．

Input: $\delta(t)$(Regular Bag Expression), *out-lab-type*$^{\lambda}_G(n)$
Output: スタートからゴールに到達すると，どの経路でも $\delta(t)$ を満たす *out-lab-type*$^{\lambda}_G(n)$ となる有向グラフ

（1） $\delta(t)$ に出現する「ラベル::型」に順序を付与する．順序は特に制約を設けない．

（2） 得られた順序に基づいて，$\delta(t)$ を Regular Expression にする．

（3） 得られた順序に合わせて，*out-lab-type*$^{\lambda}_G(n)$ を並び替えて文字列にする．$a :: t$ のような「ラベル::型」を 1 つの文字とみなす．

（4） 上記 2 と 3 で得られた Regular Expression と文字列から，[6] の手法を用いて有向グラフを生成する．

例えば，$\delta(t) \to (a :: t_1 | b :: t_1) \parallel a :: {t_2}^+$ のとき，順序を付与して $(a :: t_1 | b :: t_1) a :: {t_2}^+$ という Regular Expression にする．

一方で，$\mathit{out\text{-}lab\text{-}type}^{\lambda}_{G}(n) = |b :: t_1, a :: t_1|$ であれば，文字列 $(a :: t_1)(b :: t_1)$ に置き換える．このとき生成されるグラフを図 3 に示す．

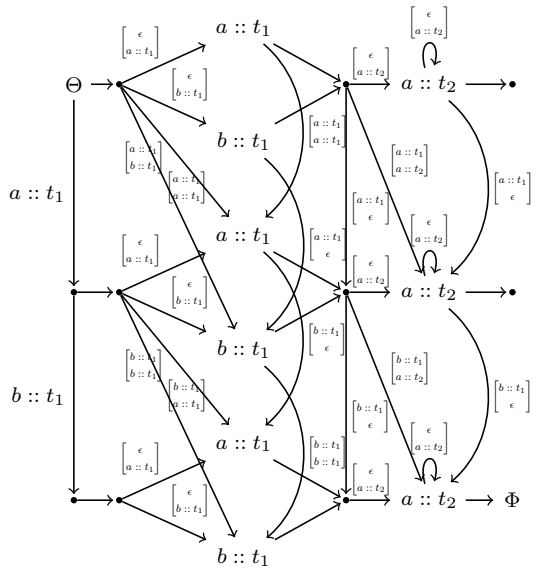


図 3　修正方法発見グラフの例

生成されたグラフについて説明する．Θ がスタート，Φ がゴールを表す．横方向のエッジはラベル型，すなわち新たなエッジの追加を意味している．$\begin{bmatrix}\epsilon\\ a :: t_1\end{bmatrix}$ では出力ラベルが a で，型が t_1 の入力ノードであるエッジを追加する．縦方向のエッジは削除を意味する．例えば，$\begin{bmatrix}a :: t_1\\ \epsilon\end{bmatrix}$ では，入力グラフ $(\mathit{out\text{-}lab\text{-}type}^{\lambda}_{G}(n))$ の $a :: t_1$ であるエッジを削除する．ただし，縦方向のエッジは場合によって置換にもなり得る．斜め方向のエッジはラベルまたは入力ノードの型，あるいはその両方の変更を意味する．$\begin{bmatrix}a :: t_1\\ a :: t_2\end{bmatrix}$ では，入力ノードの型 t_1 を型 t_2 に変更する．この斜め方向のエッジでも置換となる可能性がある．

さて，生成したグラフから最適な修正方法を導くには，エッジにコストを付与し，最小コストかつ最短となる経路を求める必要がある．まず，エッジに付与するコストは表 1 の通りに設定した．

表 1　エッジに付与するコスト

追加	1.0
削除	2.0
ラベルのみの変更	1.5
型のみの変更	2.5
ラベルと型の両方を変更	3.0
置換	0.0

ここでの「置換」はグラフに対して特に修正を加えない (同一ラベル，同一型への修正) 処理のため，コストが 0.0 となる．次に「追加」を 1.0,「削除」を 2.0，これらの組み合わせである「ラベルと型の両方を変更」は，合計値である 3.0 と設定した．ここで，削除よりも追加のコストを低く設定したのは，削除の方が低いと E が少ない型へ優先的に修正してしまうのに加えて，既にあるデータを消すよりも追加して妥当となるならばその方が有効的だと考えたからである．また「ラベルのみの変更」は，最も修正対象であるグラフに対してデータ構造の変更を伴わないため，1.5 と低く設定した．一方で「型のみの変更」は，入力ノードの型を変更するため，変更する入力ノードが出力ノードでもあった場合，そのデータ構造にも大きな影響をもたらす可能性が高い．よって，入力ノードの型を変更するよりも，エッジを変更したい型に妥当なノードに張り替える，あるいはその型に妥当なノードを追加する方がよいと考えている．以上の理由から，ラベルと型の両方の変更よりは低いコストである 2.5 と設定した．

先ほど生成したグラフのエッジに対して，ラベルを基にコストを付与する．そして，得られた重み付きグラフ上でスタートからゴールまでの最短経路を求め，そのコストを算出する．図 2 の場合，得られた経路から，$a :: t_1$ はそのままで ($a :: t_1$ に置換)，次の $b :: t_1$ をラベルと型の両方とも変更して $a :: t_2$ に修正する方法が提案される．実際には ShEx の各型と妥当でない部分グラフについてそれぞれ修正グラフを作成し，得られた修正コストのうち，最小となる型とその経路を提案する．ただし，最小となる型が複数存在する場合，全て提案する．

5　評 価 実 験

本節では妥当性検証アルゴリズム及び修正手法の評価実験について述べる．実装には，Ruby 2.6.5 を用いた．また全ての実験を，Intel Core i3-6100U @ 2.30GHz 2.30GHz CPU，8GB RAM，Windows 10 Home 64bit の環境で行った．

妥当性検証

妥当性検証アルゴリズムを評価するために，SP^{2}Bench [7]，BSBM [8] の 2 つのデータセットを用いた．これらは，RDF のベンチマークツールであり，規定された構造の RDF データを生成する．それぞれのデータセットによって生成したエッジ数 (トリプル数)，λ に格納される出力ノード数，データサイズを表 2 と表 3 に示す．SP^{2}Bench 及び BSBM には ShEx は定義されていないため，それぞれに対して ShEx を作成した．

表 2　SP^{2}Bench のデータサイズ

$\lvert E\rvert$	$\lvert V*\rvert$	size(MB)
95,768	18,584	10.1
922,241	173,154	100.9
9,053,244	1,571,136	1,008.9
18,135,502	3,152,062	2,017.8

表 3　BSBM のデータサイズ

$\lvert E\rvert$	$\lvert V*\rvert$	size(MB)
374,911	36,433	33.9
1,809,874	168,555	162.1
8,873,389	808,154	794.8
17,686,178	1,601,677	1,584.0

まず，グラフデータのデータサイズを変化させた場合に，アルゴリズムの実行時間がどのように変化するか計測した．比較対象は，3 章で示したアルゴリズム validation 及び hierarchical validation である．さらに，λ の初期定義の際に initialization アルゴリズムを用いず，λ の初期値として各出力ノードに全ての型を割り当てる場合の実行時間も計測する．結果を図 4 と図 5 に示す．

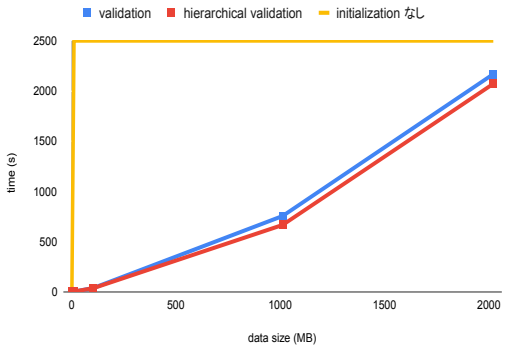


図 4　SP^{2}Bench に対する実行時間

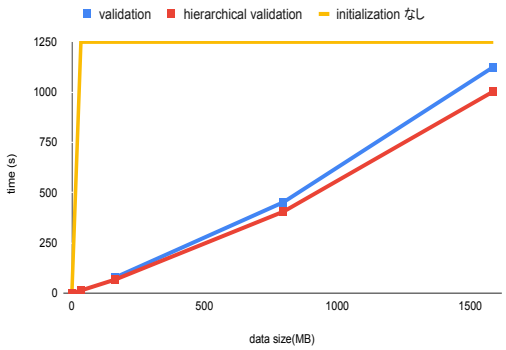


図 5　BSBM に対する実行時間

initialization アルゴリズムを用いない場合，どちらのデータセットでも実行時間は計測不能だった．よって本アルゴリズムでは，最初に出力ノードと結びつける型をある程度絞っておく必要があるといえる．一方で，本稿で提案する 2 つのアルゴリズムは，両方とも概ね線形時間で処理を完了している．また，実行時間は hierachical validation の方がやや小さくなった．通常の validation と hierarchical validation を比較すると，後者の方がファイルの読み込み回数は増えるが，fl-out-lab-$\mathit{type}_G^{\lambda}(n)$ の要素数は少ない．この結果は，出力ノードが検証している階層の型を割り当てられているかを毎回調べているため，λ が大きくなると，その時間も増えることが原因と考えている．

次に，グラフデータのデータサイズを変化させた場合に，メモリの消費量がどのように変化するか計測した．結果を図 6 と図 7 に示す．

両方のデータセットともデータサイズに対して 50%以下にメモリ消費量を抑えることができた．また，hierarchical validation の方が消費されたメモリが小さかった．hierarchical

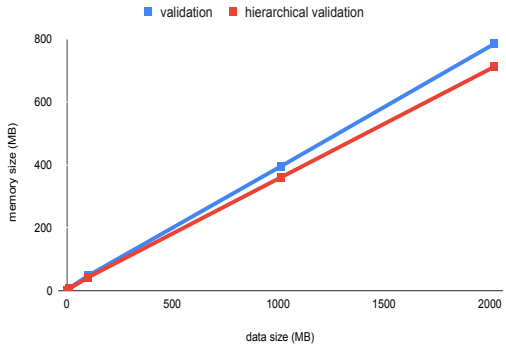


図 6　SP^{2}Bench に対するメモリ消費量

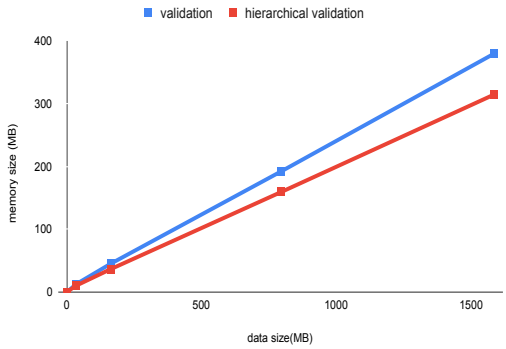


図 7　BSBM に対するメモリ消費量

validation の方が省メモリであり，実行時間もやや小さいため，サイズの大きいグラフに適していると考えられる．なお，グラフデータに対してスキーマは非常に小さくグラフサイズに影響されないので，階層を求める時間は無視できる程度に小さい．

修 正 手 法

実データに対して誤りを加えることで妥当性検証を中断し，誤り部分の型を正しく推定しその修正方法を提示できるかを確かめる．実データとして以下の教科書 LOD [9] [10] を用いた．教科書 LOD は，1992 年施行の学習指導要領以降の検定教科書を対象として、書誌事項と教科等の関連情報を LOD 化したデー他セットである．この LOD を SP^{2}Bench と同様な形式のグラフデータに変更し，評価実験を行う．ただし，空白ノードの部分は自動的にランダムなノードを作成することで，エッジが (n, a, m) の形式となるようにしている．なお，グラフのエッジ数，出力ノード数，データサイズは表 4 の通りである．

表 4　教科書 LOD のデータサイズ

$\lvert E\rvert$	$\lvert V*\rvert$	size(MB)
219,018	21,258	11.56

また，このグラフデータに妥当な ShEx を作成した．この ShEx が上記グラフデータに対して妥当であることは確認済みである．

評価実験のために，ある型に妥当なノードに対して，誤ったラベルに変更，入力型を誤った型に変更 (エッジの張りミス)，

不要なエッジの追加，エッジの削除のいずれかを行い，妥当ではない状態にする．各型に妥当であるノード 5 個に誤りを加え，正しく修正できるかを検証していく．誤りの数を増やしていき，どこまでであれば正しく修正できるかを調べる．ここで，元々妥当だった型に修正でき，なおかつその修正方法が求めているもの (加えた誤りを直す) であれば点数を 1.0 とする．型は正しく推定できたが，その修正方法が求めているものと違う場合は 0.5 点とする．全く違う型を推定した場合は 0.0 点とする．その平均値を誤りの数に対する精度と考える．表 5 に結果を示す．

表 5　誤りの数と精度

誤りの数	1	2	3	4	5
精度	0.99	0.97	0.86	0.71	0.57

概ね型は正しく推定でき，修正方法も誤りの数が 1,2 個程度であれば，著者が求めている修正方法を提案することができていた．一方で，最低出現回数が 0 回の *out-lab-type* に関しては，ひとつ前が繰り返しを許す *out-lab-type* の場合こちらを追加してしまう傾向が多々見られた．そのため，この手法ではある程度妥当な型を推定することは可能だが，適切な修正方法の提案に課題があるといえる．

また，誤りを加える際にエッジの削除のみに限定すると，最終的に t_1 や t_{11} といった型のサイズが小さいものを提案し始めることがわかっている．そのような極端な誤りの修正は難しいため，誤りの数や妥当でないノードの数に基準を設け，その基準を超えるようであれば修正せずにそのまま妥当性検証を終了する等，使い方も考慮すべきである．

5.1 ま と め

本稿では，有向グラフに対する ShEx の妥当性検証アルゴリズムを提案した．本アルゴリズムはグラフデータをメモリに載せずに検証を行い，階層の概念も導入した．評価実験の結果，提案アルゴリズムの効率性及び省メモリであることを確認することができた．一方で，階層を用いた妥当性検証では，省メモリではあるが実行時間の差がほぼ無く，課題を残す結果となった．

また，グラフデータが妥当でない場合，データ構造上誤りのあるノードに対して，最適な型とその型への修正手法も提案した．評価実験の結果，誤りが少数であれば概ね正しく提案できることを確認した．

今後の課題としては，階層化妥当性検証の実行時間を早くすることが挙げられる．現状では，グラフファイルに λ を記述し，検証している階層の型を含むか瞬時に判別できるようにする方法を考えている．この方法では，λ が主記憶に格納することができない場合でも，妥当性検証が行える可能性がある．修正手法に関しては，より型の修正方法の精度を上げること，否定等の表現にも対応できるよう拡張することを検討している．

文　献

[1] Thomas Baker, ed., “Shape Expressions (ShEx) 2.1 Primer.” http://shex.io/shex-primer/. Accessed: 2019-12-22.

[2] S. Staworko, I. Boneva, J. Labra Gayo, S. Hym, E. Prud’hommeaux, and H. Solbrig, “Complexity and expressiveness of ShEx for RDF,” 18th International Conference on Database Theory, ICDT 2015, pp.195–211, 2015.

[3] J.E.L. Gayo, E. Prud ’ hommeaux, I. Boneva, and D. Kontokostas, Validating RDF Data, Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool, 2018.

[4] A. Hemid, L. Halilaj, A. Khiat, and S. Lohmann, “RDF Doctor: A Holistic Approach for Syntax Error Detection and Correction of RDF Data,” 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2019), pp.508–516, 2019.

[5] K. Ikeda and N. Suzuki, “An Algorithm for Finding top-K Valid XPath Queries,” IPSJ Transactions on Databases, vol.7, no.2, pp.70–82, 2014.

[6] E.W. Myers and W.A. Miller, “Approximate matching of regular expressions.,” Bulletin of mathematical biology, vol.51, no.1, pp.5–37, 1989.

[7] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel, “SP^{2}Bench: a SPARQL Performance Benchmark,” Proceedings of the 25th International Conference on Data Engineering (ICDE 2009), pp.222–233, 2009.

[8] C. Bizer and A. Schultz, “The Berlin SPARQL Benchmark,” International Journal on Semantic Web and Information Systems, vol.5, pp.1–24, 2009.

[9] 江草由佳 and 高久雅生, “教科書 Linked Open Data (LOD) の構築と公開,” 情報の科学と技術, vol.68, no.7, pp.361–367, 2018.

[10] 江草由佳 and 高久雅生, “教科書 Linked Open Data (LOD).” https://jp-textbook.github.io/. Accessed: 2019-12-27.