RDFのための経路式に基づいた索引方式

的野　晃整† 　天笠　俊之† 　吉川　正俊†† 　植村　俊亮†

† 奈良先端科学技術大学院大学情報科学研究科　〒 630–0192 奈良県生駒市高山町 8916–5
†† 名古屋大学情報連携基盤センター　〒 464–8601 名古屋市千種区不老町
E-mail: †{akiyo-ma, amagasa, uemura}@is.aist-nara.ac.jp, ††yosikawa@itc.nagoya-u.ac.jp

あらまし　Semantic Web が期待される中, メタデータへの問合せの要求が高まっている. メタデータを記述するには RDF が用いられることが多く, 今後は RDF によって記述されたメタデータが大量に作成されることが予想される. そのため, RDF を高速に検索する必要がある. そこで本論文では, RDF のための索引方式を提案する. まず, 非巡回有向グラフに対する接尾辞配列の定義を行う. RDF 中の複数の関係を表現する非巡回有向グラフ構造のモデルから経路を抽出し, 定義に基づいて接尾辞配列を生成する. この索引によって検索の高速化を期待できる. また, RDF の関係には述語関係と継承関係があり両者の索引を構築することで, スキーマ情報を含む問合せを処理できる.

キーワード　Semantic Web, RDF, RDFS, 非巡回有向グラフ, 索引, 経路式, 接尾辞配列

An Indexing Scheme for RDF based on Path Expressions

Akiyoshi MATONO†, Toshiyuki AMAGASA†, Masatoshi YOSHIKAWA††, and Shunsuke UEMURA†

† Graduate School of Information Science, Nara Institute of Science and Technology (NAIST)
8916–5 Takayama-cho, Ikoma-shi, Nara 630–0192, Japan
†† Information Technology Center, Nagoya University
Furo-cho, Chikusa-ku, Nagoya-shi 464–8601, Japan
E-mail: †{akiyo-ma, amagasa, uemura}@is.aist-nara.ac.jp, ††yosikawa@itc.nagoya-u.ac.jp

Abstract　Semantic Web is expected as the framework for the next generation of the Web, and the demand for querying metadata described with RDF and RDF Schema is increasing. For this reason, it is anticipated that a large quantity of such metadata will be produced in the near future, and it is therefore important to be able to process query retrieval of RDF data efficiently. In this paper, we propose an indexing scheme for RDF data. An RDF data can be considered as four kinds of DAGs (Direct Acyclic Graphs), and we extract all possible path expressions from the DAGs. We, then, generate four kinds of suffix arrays using the path expressions. Using the indices, we can achieve efficient processing of query retrievals on RDF data containing schematic information.

Key words　Semantic Web, RDF, RDFS, Directed Acyclic Graph, Index, Path Expressions, Suffix Array

1　はじめに

次世代 Web としてその動向に大きな期待が寄せられている Semantic Web [15] [3] は, 計算機が情報の意味を理解し, 計算機同士や人と計算機間でのコミュニケーションの実現を目的に開発されている. Semactic Web にとって, RDF (Resource Description Framework) [16] と RDFS (Resource Description Framework Schema) [18] は, 基盤技術として重要な役割を担っている.

RDF はメタデータ記述の基礎を提供する. リソース間の意味を 2 項関係によって表現することができ, それらを複数用いることで, 複雑な意味を表現することができる. RDF によってメタデータを記述することで、計算機に情報の意味を付加し, より高度な処理を行わせることができる.

現在, Web 上の資源の発見には大きなコストがかかる. その理由として, 資源の意味を表現したメタデータが十分に与えられておらず, 表現の形式も統一されていないことがあげられる. RDF を用いて Web 上の資源にメタデータを記述することで, 資源の意味的な発見をすることができる [9]. 将来, Semantic Web が普及し, Web 上の情報資源に対して RDF による意味付けがなされる可能性を踏まえると, Web 上にはメタデータが膨大に増加することが予想される. その中から必要な情報を検索するためには, 高速な検索機能が必要になってくる.

これまで, RDF によって記述されたデータを処理するために, 多くの研究がなされている. しかし, それらの多くは, RDF/S の

データに対して高速に検索するという目的に関して多くは触れていない. [5], [7] および [8] では, RDF に対する問合せ言語を提案している. [1] や [2] では RDF 情報を扱うためのシステムを実装している. これらはメタデータを含む知識検索や推論などの機能の向上に重点を置いた研究である. また, [4], [11] および [12] では RDF の関係データベースへの格納や検索に関して研究を行っている. このために RDF データの有向グラフ構造から関係モデルへのマッピング手法を提案している. これらの研究は基本的に質の向上に関する研究に重きをおいており, 量的な面から RDF にアプローチする研究はなかった.

RDF は XML [17] に準拠した構文であるため, XML 検索の高速化の手法を RDF にも適用することは可能である. XML と RDF の検索とは, 両者とも意味を表現するグラフ構造中の部分グラフを発見することである. [10], [20] 及び [6] は, XML を対象としたデータベースへの格納を最適化することによって検索速度の向上を図っている. また, XML の問合せ処理の高速化のために接尾辞配列を使った研究がある [19].

しかし, これらの XML 検索の高速化の手法を RDF にそのまま適用することは実用的であるとはいえない. その理由は, 両者の表現する意味の構造が木と有向グラフで異なっているためである. 本来, 検索とは意味の構造から部分グラフを発見することが目的である. そのために, 意味の構造が異なる両者の検索は, 同一の手法を用いることは本質的ではない.

前述したように, RDF は XML に準拠しているため, 両者の構文の構造は一致している. これは, RDF が構文の構造と意味の構造が異なっていることを意味しており, RDF の処理をさらに困難にしている. つまり, 特定の要素に関連する情報を得るために, XML は要素の子孫あるいは祖先要素のみを必要とするが, RDF は文書全体を必要とする. これらの制約により, RDF 検索は XML を対象とした研究をそのまま利用することができない.

とはいえ, 前述したような XML データベースの研究を利用することは有益である. RDF に対する索引を用いることで, 対象のデータベースを特定することなく高速な検索を提供することができること考えた. まず, RDF/S から 4 種類の意味的な構造を抽出する. それらの構造は有向グラフ構造あるいは非巡回有向グラフ構造となっている. その構造を非巡回有向グラフ構造に制限する. 得られた 4 種類のモデルから経路を抽出し, その経路を経路式として列挙する. その後, それらの経路式に対して接尾辞配列で索引付けを行う手法である.

本論文の構成を次に示す. 2 章で RDF と RDFS の概要を説明する. 3 章で非巡回有向グラフモデルの経路に対する接尾辞配列を定義する. その後, 4 章で例を挙げながら, RDF/S のモデルからそれぞれ独立な関係を持つ 4 種類の経路を複数抽出し, 5 章で, 定義に従って, それらの経路から接尾辞配列を生成する. 6 章で本研究の提案手法の実装と評価を行い, 最後にまとめる.

2 RDF/S の概要

RDF はメタデータのための枠組みである. RDF はメタデータの相互運用や機械が理解可能な意味の記述, 正確な資源の発見などの多くの機能を提供することが可能である.

```
<rdf:RDF>
  <rdf:Description about="www.matono.net/paper">
     <s:authored>Akiyoshi MATONO</s:authored>
  </rdf:Description>
</rdf:RDF>
```

図 1　RDF/XML 表現の例

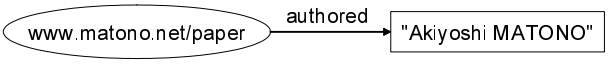

図 2　文 (statement) の例

RDF の 2 項関係を表現できる最小単位の表現を複数用いることで, 複雑な情報を表現することができる. この最小単位を文 (rdf:statement) と呼ばれる. 本論文では, この関係を述語関係と呼ぶ. 文を用いることで, たとえば, “This paper is authored by Akiyoshi MATONO.” という意味を表現することができる. 主語 (rdf:subject, ここでは “This paper”), 述語 (rdf:predicate, “is authored by”), および目的語 (rdf:object, “Akiyoshi MATONO”) によって成り立っている. このように, 3 つの部分によって構成されているためトリプルとも呼ばれることがある.

RDF は XML [17] に準拠した仕様となっており, この例を RDF で表現すると図 1 のようになる.

ここで rdf は名前空間 “http://www.w3c.org/2000/01/rdf-schema#” を参照するための接頭辞である. s は “authored” のプロパティを定義したスキーマの名前空間接頭辞である. 後述する rdfs は名前空間 “http://www.w3c.org/1999/02/22-rdf-syntax-ns#” を参照するための接頭辞である. また, 本論文では, RDF と RDFS をあわせて RDF/S と呼ぶ.

例のモデルを図で表すと図 2 のようになる. このように, RDF/S の文が表現する意味のモデルは主語と目的語を頂点とし, 述語を有向辺とする有向グラフの構造となる.

この例では, 主語は rdfs:Class の子クラス (クラス) のインスタンス, 述語は rdf:Property の子クラス (プロパティ), 目的語は rdfs:Literal の子クラス (リテラル) のインスタンスである. インスタンスとは, RDF で記述された rdfs:Resource を継承するクラスやプロパティの実体を指す. rdfs:Class や rdf:Property, rdfs:Literal は rdfs:Resource の子クラスである. つまり, RDF/S はすべて rdfs:Resource のメンバ (リソース) によって記述されている.

RDFS は RDF で利用するリソースのタイプ (クラスやプロパティなど) を定義するための仕様である. つまり, クラス間やプロパティ間の継承関係や, クラス間の述語関係, プロパティのドメインや対象の範囲などを定義することができる. 図 1 の場合, RDFS によって “authored” のプロパティを定義している. s はその RDFS の名前空間を指し示すための接頭辞である.

RDF の表現は RDFS で定義されたスキーマのインスタンスである. また, RDFS での文の型の定義は図 1 とは異なり, 主語となるクラスをドメイン, 目的語となるクラスを範囲として, そ

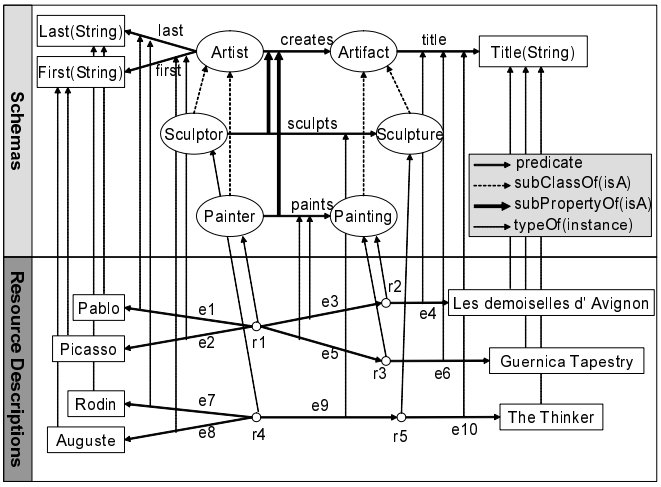

図 3　RDF/S の例

れらを含むプロパティを定義する.

RDF では, 複数の文を用いて複雑な意味を表現することが可能である. 図 3 に少し複雑な RDF/S の例を示す. 図の上部では, RDFS によってクラスやプロパティ間の述語関係や継承関係を定義している. 下部は RDF によって, RDFS で定義されたクラスやプロパティのインスタンスを記述している.

RDF/S で表現可能な関係は, 1) RDFS でのクラス間の述語関係, 2) RDFS でのクラス間の継承関係, 3) RDFS でのプロパティ間の継承関係, 4) RDF でのリソース間の述語関係, 5) RDF の各リソースがどのクラス (プロパティ) のインスタンスであるかを表した関係である. 2), 3) の継承関係を表現するモデルはクラスを頂点とし, ラベルの無い有向辺をもつ非巡回有向グラフとなり, 1), 4) の述語関係を表現するモデルはクラスやリソースがグラフの頂点, プロパティを有向辺とする有向グラフとなる. 5) のインスタンスは, 2), 3) の継承関係のモデルの葉にマッピングできる. それぞれのモデルは独立して表現できる.

3　非巡回有向グラフに対する接尾辞配列

RDF/S と XML の検索とは, 両者ともグラフ構造中の部分グラフを発見することである. XML を対象とした問合せ処理の高速化のために接尾辞配列を用いた研究 [19] があるが, これを RDF に適用することは難しい. その理由として, 両者が表現する意味の構造の違いにある. また, RDF は意味の構造と構文の構造が異なっていることもその理由として挙げられる.

本論文では, 対象のデータベースを特定することなく高速な検索を提供することができる RDF のための索引を提案する. 提案する索引は, 有向グラフを対象にしているため, 経路に基づいた索引である. 具体的には, RDF/S のモデルから得られる経路式の接尾辞配列を用いる方式である.

本来, 接尾辞配列はテキストに対する索引方式であり, 有向グラフに対するものではない. RDF/S のデータ構造は基本的に有向グラフであるため, 有向グラフに対する接尾辞配列を定義する必要がある. しかし, 有向グラフはループを含むため, データ構造は複雑で, 一般的に容易に扱うことはできない. また, RDF/S の継承関係のみのデータ構造は非巡回有向グラフである. このため, 本論文では RDF/S のデータ構造を非巡回有向グラフに限定し, 非巡回有向グラフに対する接尾辞配列を定義する.

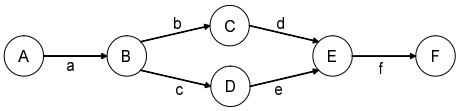

図 4　非巡回有向グラフの例

	1	2	3	4	5	6	7	8	9
1	A	a	B	b	C	d	E	f	F
2	A	a	B	c	D	e	E	f	F

図 5　経路の接尾辞

しかし, 実際の RDF/S ではループを含む情報は多く存在する. 今後研究を進めていく上でそれらを無視することはできないので, それは今後の課題とする.

非巡回有向グラフに対する接尾辞配列では経路を対象にする. 非巡回有向グラフに対する接尾辞配列を以下のように定義する.

［**定義** 1］（非巡回有向グラフに対する接尾辞配列） 非巡回有向グラフ G は頂点有限集合 $V(G)$, 有向辺有限集合 $E(G)$ から構成される. $E(G)$ は $V(G)$ の相異なる 2 要素の順序対を要素とする. G は閉路を含まない.

点 u から点 v へ向かう有向辺を $e = (u, v)$ のように表し、このときの点 u, v をそれぞれ始点, 終点と呼ぶ. また, 入次数が 0 である頂点集合 $L \subset V(G)$ の要素を根と呼ぶ. 出次数が 0 である頂点集合 $M \subset V(G)$ の要素を葉と呼ぶ.

G において, 以下の条件の時, $p_t = s_{t,1}.s_{t,2}.\cdots.s_{t,k_t-1}.s_{t,k_t}$ $(1 \leq t)$ は G の根 $s_{t,1} \in L$ と葉 $s_{t,k_t} \in M$ を結ぶ, t 番目の長さ k_t の経路であるという.

- $s_{t,2h-1} \in V(G)$ $(1 \leq h \leq \frac{k_t+1}{2})$
- $s_{t,2h} = (s_{t,2h-1}, s_{t,2h+1}) \in E(G)$ $(1 \leq h \leq \frac{k_t-1}{2})$

j 番目の長さ k_j の経路 $p_j = s_{j,1}.s_{j,2}.\cdots.s_{j,k_j}$ において, p_j の i 番目の接尾辞を $S_{j,i} = s_{j,i}.s_{j,i+1}.\cdots.s_{j,k_j}$ とし, その索引点を $a_{j,i} = [j, i]$ という二次元配列とする.

経路 p_j の接尾辞配列 $S(p_j)$ とは, すべての接尾辞を辞書順に並べ替えた, 長さ k_j の索引点の配列のことである.

G の接尾辞配列 $S(G)$ とは, 根 $\{u|u \in L\}$ に連結した葉 $\{v|v \in M\}$ を結ぶ, すべての経路 $P(G)$ の接尾辞を辞書順に並べ替え, 重複部分を削除した索引点の配列のことである. □

非巡回有向グラフの例を図 4 に挙げ, それに対する接尾辞配列を説明する. 得られる経路は “A.a.B.b.C.d.E.f.F” と “A.a.B.c.D.e.E.f.F” の二つである. それらの経路に対し, 図 5 のように接尾辞に索引点を二次元配列で与え, 図 6 のように辞書順に並べ替え, 重複経路を削除する. 結果, 図 4 の接尾辞配列は [1,1] [2,1] [1,3] [2,3] [1,5] [2,5] [1,7] [1,9] [1,2] [2,2] [1,4] [2,4] [1,6] [2,6] [1,8] となる.

4　RDF/S の経路式

経路式とは, RDF/S で表現されるそれぞれ独立した関係を非巡回有向グラフ構造のモデルで表し, その経路を式で表現したものである. 例えば, 図 2 の RDF の例を経路式で表現すると以

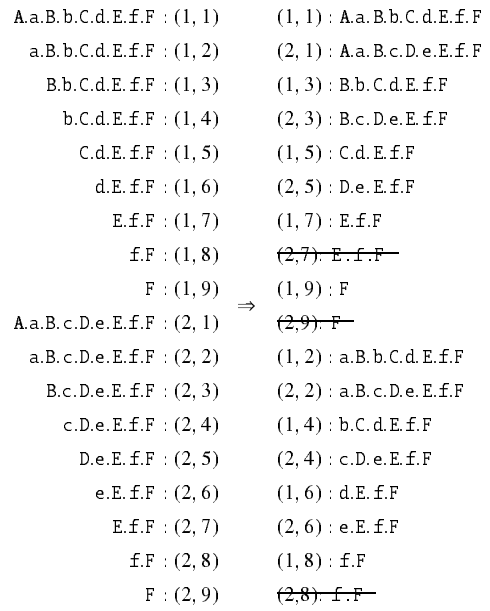

図 6　接尾辞のソートと削除

下のようになる.

#www.matono.net/paper > +authored > "AkiyoshiMATONO"

このように述語関係の経路式は, クラスやリテラルを頂点とし, プロパティを有向辺としたモデルの根から葉までの経路を表すものである.

我々が提案する経路式は, 次のような構文で記述する. 経路式のデリミタは ">" とする. rdfs:Literal を継承したインスタンスは """ で囲み, クラスやプロパティと区別をするために rdfs:Class のメンバとなるクラスの先頭には "#" を, rdf:Property のメンバとなるプロパティの先頭に "+" をつける. また, この例では記述されていないが, 継承関係の経路式の時インスタンスは先頭に "$" をつける. ここでの特殊文字が RDF/S 文書中に記述されている場合は, 経路式を生成する際には XML の実体参照に変換する.

これらのクラスにのみ, 接頭辞をつける理由を次に示す. RDF/S の仕様で定義されたクラスは, ここで挙げるクラス以外にもあるが, RDF/S を記述する上で主に rdfs:Class, rdf:Property のメンバを用いることが多い. また, rdfs:Class と rdf:Property は意味的な差が大きく, それぞれを継承するメンバは大きく異なった使われ方をする. そのため, それらを識別するための前述したような接頭辞をつけた. また, リテラルは文字列であるため, 空白などの特殊文字を含む可能性があることを考慮した.

本論文で提案する経路式は, 以下の 4 種類である. これらはクラスやプロパティ間の述語関係と継承関係から得られる非巡回有向グラフのそれぞれ独立したモデルから抽出する.

（1） **スキーマ述語関係経路式**

RDFS で定義されたクラス, プロパティ間の述語関係のモデルから得られる経路式.

（2） **インスタンス述語関係経路式**

RDF で記述されたインスタンス間の述語関係を表現したモデル

```
1:#Artist>+last>#Last
2:#Artist>+first>#First
3:#Artist>+creates>#Artifact>+title>#Title
4:#Sculptor>+sculpts>#Sculpture
5:#Painter>+paints>#Painting
```

図 7　図 3 の例のスキーマ述語関係経路式

```
1:#r1>+e1>"Pablo"
2:#r1>+e2>"Picasso"
3:#r1>+e3>#r2>+e4>"Les demoiselles d' Avignon"
4:#r1>+e5>#r3>+e6>"Guernica"
5:#r4>+e7>"Rodin"
6:#r4>+e8>"Auguste"
7:#r4>+e9>#r5>+e10>"The Thinker"
```

図 8　図 3 の例のインスタンス述語関係経路式

から得られる経路式. 前述した経路式の例はこの経路式である.

（3） **クラス継承関係経路式**

RDFS で定義されたクラス間の継承関係のモデルの葉に RDF のクラスのインスタンスをマージしたモデルから得られる経路式.

（4） **プロパティ継承関係経路式**

RDFS で定義されたプロパティ間の継承関係のモデルの葉に RDF のプロパティのインスタンスをマージしたモデルから得られる経路式.

一般的な RDFS 文書は, RDFS の仕様で定義されたクラスを利用して記述される. すなわち, RDFS の仕様はスキーマ定義のためのスキーマと言える. しかし, RDFS を用いないで RDF を記述する稀なケースもありうる. このとき, 2) 以外は本来, スキーマのない RDF からは作成することができないが, RDFS の仕様自体をスキーマとして扱うことで, すべての関係の経路式を表現できる.

1) と 2) の経路式では "#a>+b>#c" は "a の b は c である" と読みかえることができる. 1) と 2) の関係の意味は等しいため, 総称して述語関係経路式と呼ぶ. また, 3) と 4) の経路式で "a>b>$c" は, "a" は "b" の親クラス, "c" は "b" のインスタンスであることを意味している. 3) と 4) での関係の意味は等しいため, 継承関係経路式と呼ぶ. また, 継承関係経路式では 図 9 や図 10 のように、経路の葉にはインスタンスが記述されることで, そのインスタンスがどのクラス (プロパティ) を継承したインスタンスであるかを判定することが可能になる.

前述した図 3 の例のすべてのスキーマ述語関係経路式を図 7 に, インスタンス述語関係経路式を図 8 に, クラス継承関係経路式を図 9 に, プロパティ継承関係経路式を図 10 に示す.

また, 非巡回有向グラフから経路式を生成するアルゴリズムを図 11 に示す. ここで生成する経路式は述語関係か継承関係かは限定しない. 非巡回有向グラフの根は入次数が 0 である頂点で, 葉は出次数が 0 である頂点である.

5　RDF/S の接尾辞配列

本章では経路式から接尾辞配列を生成する過程を具体的に説

```
1:#Artist>#Painter>$r1
2:#Artist>#Sculptor>$r4
3:#Artifact>#Painting>$r2
4:#Artifact>#Painting>$r3
5:#Artifact>#Sculpture>$r5
6:#Last>"Pablo"
7:#Last>"Rodin"
8:#First>"Picasso"
9:#First>"Auguste"
10:#Title>"Les demoiselles d' Avignon"
11:#Title>"Guernica"
12:#Title>"The Thinker"
```

図 9　図 3 の例のクラス継承関係経路式

```
1:+last>$e1
2:+first>$e2
3:+last>$e7
4:+first>$e8
5:+creates>+paints>$e3
6:+creates>+paints>$e5
7:+creates>+sculpts>$e9
8:+title>$e4
9:+title>$e6
10:+title>$e10
```

図 10　図 3 の例のプロパティ継承関係経路式

明する. その後, 生成した接尾辞配列を用いて検索する過程を説明する.

図 3 に示した RDF/S からスキーマ述語関係経路式の接尾辞配列を生成する. 述語関係経路式, 継承関係経路式でも接尾辞配列を生成する過程は同じである. 定義 1 で定めたように, 対象の関係から非巡回有向グラフ構造のモデルを生成し, そのモデルのすべての経路式を列挙する (図 7). そして, 各経路式の接尾辞に対して, 索引点を与え (図 12), 全体を辞書順に並べ替え, 重複要素を削除する (図 13). 図 12 のように各経路を接尾辞ごとに, 索引点を与える. 最後に, 接尾辞をキーとして辞書順に並べ替えた索引点 [3,3] [3,1] [2,1] [1,1] [2,3] [1,3] [5,1] [5,3] [4,1] [4,3] [3,5] [3,2] [2,2] [1,2] [5,2] [4,2] [3,4] が RDF/S の接尾辞配列となる.

接尾辞配列を用いて問合せに対する解を得るには, 二分探索あるいは B 木を用いて検索する. そのため, 計算量は高々 $O(log_2(n+1))$ となる. このとき n は接尾辞数である.

6　性能と評価

本章では, 提案している索引を実装し, その評価を行う.

実験の流れを説明する. 1) RDF/S に対する代表的な問合せを決定する. 2) 対象のデータベースを決定する. 3) 格納する RDF/S データを決定する. 4) RDF/S を関係データベースに格納し, RDF/S のサイズを計測する. 6) その後, 具体的な問合せを決定する. ここまでが準備段階である. 7) 格納されたデータから索引を生成する. 8) 格納された索引のサイズ, 行数を計測し, 9) 索引を使う場合と使わない場合で問合せの処理時間を計測する.

```
var roots : 頂点の集合
var stack : Stack
starts := 経路の根の集合
foreach start (roots) begin
   start := start
   createPath(start)
end
function createPath(start : 頂点) : Void
var end : 頂点
var edges : 有向辺の集合
var triple : (頂点, 有向辺, 頂点) の組
begin
  if(start != nil) then
     edges := start(始点) に接続する有向辺の集合
     foreach edge(edges) begin
        end := edge に接続する終点
        triple := (start, edge, end) の組
        stack に triple を push
        stack を元に経路式を生成
        createPath(end)
        stack を pop
     end
  end
end
```

図 11　経路式生成アルゴリズム

```
(1,1):  #Artist>+last>#Last
(1,2):  +last>#Last
(1,3):  #Last
(2,1):  #Artist>+first>#First
(2,2):  +first>#First
(2,3):  #First
(3,1):  #Artist>+creates>#Artifact>+title>#Title
(3,2):  +creates>#Artifact>+title>#Title
(3,3):  #Artifact>+title>#Title
(3,4):  +title>#Title
(3,5):  #Title
(4,1):  #Sculptor>+sculpts>#Sculpture
(4,2):  +sculpts>#Sculpture
(4,3):  #Sculpture
(5,1):  #Painter>+paints>#Painting
(5,2):  +paints>#Painting
(5,3):  #Painting
```

図 12　索引点の付与

10) 最後に考察を行う.

尚, 経路式と接尾辞配列の生成に要する処理時間は索引付けを行う対象のデータの構造に大きく依存するため, 今回は評価の対象としない.

6.1　実験環境

実験に用いるマシンの性能は, CPU が Athlon 1.1GHz, メモリが 768MBytes である.

RDF/S に対する代表的な問合せを表 1 に示す. 表 1 の上部は

```
(3,3): #Artifact>+title>#Title
(3,1): #Artist>+creates>#Artifact>+title>#Title
(2,1): #Artist>+first>#First
(1,1): #Artist>+last>#Last
(2,3): #First
(1,3): #Last
(5,1): #Painter>+paints>#Painting
(5,3): #Painting
(4,1): #Sculptor>+sculpts>#Sculpture
(4,3): #Sculpture
(3,5): #Title
(3,2): +creates>#Artifact>+title>#Title
(2,2): +first>#First
(1,2): +last>#Last
(5,2): +paints>#Painting
(4,2): +sculpts>#Sculpture
(3,4): +title>#Title
```

図 13　接尾辞の一覧

表 1　RDF/S に対する代表的な問合せ

スキーマ問合せ	
Q1	あるプロパティの domain となるクラスの検索
Q2	あるプロパティの range となるクラスの検索
Q3	あるクラスを domain とした時のプロパティの検索
Q4	あるクラスを range とした時のプロパティの検索
Q5	あるクラスの子クラスの検索
Q6	あるクラスの子孫クラスの検索
Q7	あるクラスの親クラスの検索
Q8	あるクラスの祖先クラスの検索
Q9	あるプロパティの子プロパティの検索
Q10	あるプロパティの子孫プロパティの検索
Q11	あるプロパティの親プロパティの検索
Q12	あるプロパティの祖先プロパティの検索
インスタンス問合せ	
Q13	あるリソースを主語とした時, 述語となるリソースの検索
Q14	あるリソースを主語とした時, 目的語となるリソースの検索
Q15	あるリソースを目的語とした時, 述語となるリソースの検索
Q16	あるリソースを目的語とした時, 主語となるリソースの検索
Q17	あるリソースを述語とした時, 主語となるリソースの検索
Q18	あるリソースを述語とした時, 目的語となるリソースの検索
インスタンスとスキーマが混合した問合せ	
Q19	あるクラスのインスタンスの検索
Q20	あるプロパティのインスタンスの検索
Q21	あるリソース (クラスのインスタンス) のクラスの検索
Q22	あるリソース (プロパティのインスタンス) のプロパティの検索

スキーマに対する問合せで, 中部はインスタンスに対する問合せ, 下部はインスタンスとスキーマのどちらともの情報が必要な問合せである.

提案手法を評価する際にはこれらの問合せを用いるが, 本提案手法ですべての問合せに利用できるわけではない. すなわち, 本論文で提案した経路による索引では, Q2 や Q5, Q6, Q13, Q15 などのような順方向の問合せに関しては有効に高速化できると予想される. しかし, Q1 や Q7, Q8, Q16, Q17 などのような逆方向の問合せに対しては利用することができない. 例えば, 「"Artist" が "create" するものは何か」という問合せには有利だが, 「"Artifact" は何に "create" されるのか」という問合せには不利である. [19] では, このような問題を解決するために逆経路による索引を提案している. 今後, 本研究でも逆経路を導入することを検討する予定であるが, 本実験ではこのような逆方向の問合せに関しては評価していない.

表 2　Word Net の RDF/S 文書の詳細

RDF/S 文書数	5
スキーマ文書数	1
インスタンス文書数	4
RDF/S 文書サイズ 合計 (KB)	2,235
スキーマ文書のサイズ (KB)	4
スキーマ文書のサイズの合計 (KB)	2,231
インスタンス文書の要素, 属性の数の合計	68,108
クラス数	6
プロパティ数	5
クラスのインスタンス数	18,822
インスタンス文書中のプロパティ数	24,726
DB 全体のサイズ (KB)	12,468
クラス関連のテーブルサイズ (KB)	672
クラス関連のテーブルの行数	21,883
プロパティ関連のテーブルのサイズ (KB)	3,288
プロパティ関連のテーブルの行数	24,695

実験では, RDFSuite [1] の関係データベースに対して索引を生成した. 本索引を利用した場合と利用していない場合を比較して評価を行う. RDFSuite は PostgreSQL7.1.3 を用いて実装している. また, RDFSuite は GenRepr と SpecRepr の 2 種類のスキーマを持っている. GenRepr は, Resources と Triples という 2 種類のテーブルを持っており, 前者は資源とその識別子を格納し, 後者は資源間の文を表現するためのテーブルである. SpecRepr は XML を対象とした関係データベースへの格納の研究である [6], [14] の attribute-based アプローチに似ている. 実験では, 比較的高速な検索を実現している SpecRepr を対象のデータベースとして利用した.

今回, 評価に利用したデータは Word Net Project [13] の RDF/S データである. Word Net Project はクラスの階層構造のある単語を定義した語彙データベースの作成を行うプロジェクトである. 現在, Word Net の RDF/S 文書のサイズは約 33Mbytes であるが, 本実験では時間的制約のため, 約 2Mbytes の大きさに削除して実験を行った. Word Net の RDF/S 文書のサイズの詳細を表 2 に示す. 対象の RDF データにはプロパティの継承関係を持っていないデータであるため, プロパティの継承関係を用いた問合せ (Q9 から Q12) に関しては実験できない. また, インスタンスの述語関係は, 述語としてプロパティのインスタンスを用いおらず, プロパティ自体を用いているため, プロパティのインスタンスを求める問合せ (Q20, Q22) も実験できない.

問合せは RDFSuite は C で PostgreSQL の関数を作成し, データベースとのインターフェイスは C++によって実装している.

表 4　索引データのサイズ

	サイズ (KB)	行数
索引全体	19,328	182,496
索引全体の経路情報	13,688	52,481
索引全体の接尾辞配列	5,640	130,015
スキーマ述語関係の経路情報	8	10
スキーマ述語関係の接尾辞配列	8	21
インスタンス述語関係の経路情報	12,712	45,454
インスタンス述語関係の接尾辞配列	4,360	100,737
クラス継承関係の経路情報	968	7,017
クラス継承関係の接尾辞配列	1,272	29,257
プロパティ継承関係の経路情報	0	0
プロパティ継承関係の接尾辞配列	0	0

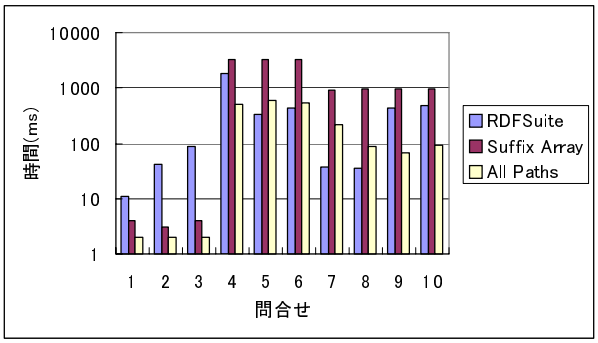

図 14　処 理 時 間

提案手法は接尾辞配列に対する二分探索を PL/pgSQL で PostgreSQL 内に関数を生成し, インターフェイスは Java1.4 で実装した.

評価する具体的な問合せの経路式を表 3 に示す. ここで, "&wn;" は Word Net の名前空間を参照するための実体参照である. 問 1 から問 3 まではスキーマ述語関係に基づいた問合せ, 問 4 から問 6 まではインスタンス述語関係に基づいた問合せ, 問 7 から問 10 まではクラス継承関係に基づいた問合せである. 前述したように, プロパティ継承関係は対象の RDF/S にないため実験できない. 括弧の中はその問合せが表 1 に示したどの問合せに該当するかを表している.

6.2 実 験 結 果

索引を生成した. そのサイズを表 4 に示す. 索引全体のサイズとそれぞれの索引のサイズの合計が異なるのは, 索引のための索引が PostgreSQL によって生成されているためであると考えられる. この結果から, RDFSuite のデータに比べ索引データは非常に大きい事が分かる.

図 14 に問合せの処理時間の実験結果を示す. 図 14 は, 表 3 で示した問合せを異なる条件の下で処理した結果である. 縦軸は時間 (ms), 横軸は問合せの種類である. また, 時間軸は対数となっている. RDFSuite は提案手法の索引を用いない場合の処理時間, Suffix Array は提案手法を用いた場合の処理時間, All Paths は提案手法を拡張した場合の処理時間である. 提案手法の拡張に関しては後述する.

提案手法を用いない場合と用いた場合を比較すると, 問 1 から問 3 までのスキーマ述語関係に対する問合せに関しては, 提案手法を用いた場合の方がよい結果を得ている. しかし, そのほかの問合せに関しては提案手法を用いない場合の方がよい. 問 1 から問 3 までの問合せは, 他の問合せに比べ, 関係を表現する接尾辞のデータ量が小さいことが要因であると考えることができる. この結果により, 提案手法はデータ量の小さな場合に限り有効であることが分かる.

問 4 と問 5 での RDFSuite による検索の速度を比較すると, 問 4 が遅い. hyponymOf プロパティのテーブルの行数は 3753 行で, wordForm プロパティのテーブルの行数は 12290 行であるため, 同じ条件下であれば, 問 4 の方が早いはずである. その理由を説明する. RDFSuite のスキーマのプロパティに関するテーブル構造は, そのドメインと範囲の値を保持している. それらのテーブルには, ドメインには PostgreSQL の BTree の索引が張られているが範囲には索引が張られていないためであると考えられる.

また, 本提案手法は問 1 から問 3, 問 4 から問 6, 問 7 から問 10 の処理時間が近いことが分かる. それぞれの検索の対象である接尾辞配列が同じであるためで, その行数に依存しているためである事が分かる.

他の問合せでよい結果が得られなかった理由としては, 接尾辞配列から経路式を特定する点がボトルネックとなっていると考えることができる. その理由を次に示す.

提案手法は, 経路式と接尾辞配列を異なるテーブルで格納して, 接尾辞配列から接尾辞としての経路式の部分を求める手法である. それに対し, すべての接尾辞を単一のテーブルに格納することで, 接尾辞配列から特定の接尾辞を得る処理を省くことができる. この手法の処理時間を図 14 の All Paths に示す.

この結果より, 本来の提案手法より高速化を図ることができた. つまり, 本来の提案手法では接尾辞配列から接尾辞のマッピングの処理がボトルネックとなっていることが分かる. この拡張した手法の問題点としては, データ量が増加することが挙げられる.

問 5 から問 8 に関してはまだ RDFSuite の方が高速である. 問 5 と問 6 の場合, この手法では検索対象の接尾辞の行数が 100737 行であるのに対し, RDFSuite は, 問 5 で 12301 行, 問 6 で 16054 行のデータから検索している. 検索対象のデータ量の差が現れていると考えられる. また, 問 7 と問 8 でも, 提案手法の場合は 29257 行から検索しているのに対し, RDFSuite の場合, 6 行から検索している. ただし, 継承の数だけ表結合を必要としている. RDFSuite のデータベースのスキーマはクラス間の継承関係を表現するための表があり, そのため高速に検索することができている.

問 9 と問 10 ではクラスのインスタンスの検索を行う問合せである. このようなデータ量の多いインスタンスの検索には, RDFSuite では処理時間が多くかかることが分かる. これは, RDFSuite は特定のクラスのインスタンスの量が少なくて済むようなデータには向いていることを意味している. つまり, RDFS によって複雑に構造化されていない RDF/S には向いていない. 本提案手法は, データの量に左右されるが, RDFS によって定義される構造には依存せず, 検索の速度はある程度予想できると思われる.

表 3　評価実験に用いた問合せ

	スキーマ述語関係	
問 1	+glossaryEntry>#	プロパティの range 検索 (Q2)
問 2	#LexicalConcept>+	クラスのプロパティ検索 (Q3)
問 3	#LexicalConcept>+antonymOf>#LexicalConcept>+hyponymOf>#LexicalConcept>+	長い経路式 (Q2, Q3)
	インスタンス述語関係	
問 4	+hyponymOf>#	目的語の検索 (Q15)
問 5	#&wn;300002062>+wordForm>#	文の検索 (Q13, Q14, Q18)
問 6	#&wn;300003694>+similarTo>#&wn;300004064>+wordForm>#	長い経路式 (Q13, Q14, Q18)
	クラス継承関係	
問 7	#LexicalConcept>#	子クラスの検索 (Q5)
問 8	#LexicalConcept>#Adjective>#	孫クラスの検索 (Q5, Q6)
問 9	#Adjective>$	インスタンスの検索 (Q19)
問 10	#Resource#LexicalConcept>#Adjective>#AdjectiveSatellite>$	長い経路式 (Q5, Q6, Q19)

7 まとめ

Semantic Web が次世代 Web として期待されている中, メタデータ記述の基礎である RDF/S の増加が予想される. 同時に RDF/S の高速な検索の要求が高まっている. RDF/S の構文は XML に準拠しているが, その意味の構造は有向グラフである. そのため, XML 関連の技術の多くはそのまま利用することができない. 本論文では, RDF/S に対する問合せを高速化するための索引を提案し, 実装評価した. これによって, 対象のデータベースを特定することなく RDF/S を格納でき, 意味的な問合せを高速に行うことが可能である.

RDF/S が表現する意味の構造は, それぞれ独立した 4 種類の関係から成っている. それらの構造は有向グラフであるが, 本論文では非巡回有向グラフに制限した. 4 種類の関係から非巡回有向グラフのモデルを生成し, そのモデルから非巡回有向グラフに対する接尾辞配列の定義に沿って, 経路を抽出し, その経路から接尾辞配列を得る手法である.

その結果, 接尾辞配列から接尾辞を特定する部分の処理が問題となっていることが分かった. また, 本提案手法では全体のデータ量に依存し, スキーマに依存しないことが分かった. この問題点を踏まえ改良を行うことを今後の課題としたい.

また, すべての RDF/S に対応できてないという問題が残されている. 具体的には, ループを含む有向グラフに対応する点や名前空間や rdfs:Bag, rdfs:Seq などのコンテナクラスなどである. まだ, 多くの課題が残されているため, これらについて早急に議論したい.

文　献

[1] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The RDFSuite: Managing Voluminous RDF Description Bases, 2000.

[2] Dave J. Beckett. The design and implementation of the redland RDF application framework. In *World Wide Web*, pages 449–456, 2001.

[3] Tim Berners-Lee. What the Semantic Web can represent. http://www.w3.org/DesignIssues/RDFnot.html, September 1998.

[4] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: An Architecture for Storing and Querying RDF Data and Schema Information, 2001.

[5] Stefan Decker, Dan Brickley, Janne Saarela, and Jurgen Angele. A Query and Inference Service for RDF.

[6] Daniela Florescu and Donald Kossmann. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. Technical report, INRIA, 1999.

[7] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. RQL: a declarative query language for RDF. In *Proceedings of the eleventh international conference on World Wide Web*, pages 592–603. ACM Press, 2002.

[8] Ashok Malhotra and Neel Sundaresan. RDF Query Specification. http://www.w3.org/TandS/QL/QL98/pp/rdfquery.html.

[9] Eric Miller, Paul Miller, and Dan Brickley. Guidance on expressing the Dublin Core within the Resource Description Framework (RDF). http://www.ukoln.ac.uk/metadata/resources/dc/datamodel/WD-dc-rdf/, July 1999.

[10] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 302–314, 1999.

[11] Storing RDF in a relational database. http://www-db.stanford.edu/~melnik/rdf/db.html.

[12] Survey of RDF/Triple Data Stores. http://www.w3.org/2001/05/rdf-ds/DataStore.

[13] the Cognitive Science Laboratory at Princeton University. Word Net. http://www.cogsci.princeton.edu/ wn/.

[14] F. Tian, D. DeWitt, J. Chen, and C. Zhang. The design and performance evaluation of alternative xml storage strategies, 2000.

[15] World Wide Web Consortium. Semantic Web. http://www.w3c.org/2001/sw/.

[16] World Wide Web Consortium. Resource Description Framework(RDF) Model and Syntax Specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, Febuary 1999. W3C Recommendation 22 February 1999.

[17] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/REC-xml, October 2000. W3C Recommendation 6 October 2000.

[18] World Wide Web Consortium. Resource Description Framework(RDF) Schema Specification 1.0. http://www.w3.org/TR/2000/CR-rdf-schema-20000327/, March 2000. W3C Candidate Recommendation 27 March 2000.

[19] Yohei Yamamoto, Masatoshi Yoshikawa, and Shunsuke Umeura. On Indices for XML Documents with Namespaces. In *Conference Proceedings of Markup Technologies '99, GCA, Philadelphia, U.S.A.*, 1999.

[20] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, and Shunsuke Uemura. XRel: a path-based approach to storage and retrieval of XML documents using relational databases. *ACM Transactions on Internet Technology (TOIT)*, 1(1):110–141, 2001.