NAIST-IS-DD0261026

博士論文

経路式に基づく RDF データの格納と検索に関する研究

的野 晃整

2005 年 2 月 3 日

奈良先端科学技術大学院大学
情報科学研究科 情報生命科学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である。

的野 晃整

審査委員：

植村 俊亮 教授　（主指導教員）
関 浩之 教授　（指導教員）
宮崎 純 助教授　（指導教員）
松本 裕治 教授　（委員）
吉川 正俊 教授　（委員，名古屋大学）

経路式に基づく RDF データの格納と検索に関する研究*

的野 晃整

内容梗概

本論文では，大規模な Resource Description Framework (RDF) データを効率的に格納，検索するための経路式に基づいた手法を提案する． RDF とは，次世代 Web として期待が寄せられている Semantic Web 実現のために提案されたメタデータ記述のための枠組みである． RDF は 2 項関係を表現する文を基本単位として構成しているために， RDF データはコンテンツを頂点とした有向グラフ構造となる．

今日， RDF はさまざまなメタデータを記述するために利用されはじめている．大規模な RDF データも登場してきており，今後， RDF の普及に伴って増加することが予想できる．このため，大量の RDF データを高速に処理することのできる RDF データベースや索引手法が重要である．

これまで，多くの RDF データベースが提案されている．これらの最大の問題点として，経路式に基づく問合せの検索処理の計算コストが高い点が挙げられる．従来の RDF データベースは，RDF データを文に分割して格納する手法を採用しているために，経路式に基づく問合せ処理において，経路長に応じた結合演算が必要になり，検索性能の低下につながる．もう一つの問題点として，スキーマデータの扱いに関する問題がある．従来の RDF データベースには，スキーマデータを扱うことを前提にして設計されたものと，そうでないものとがある．前者は，スキーマデータを持たない RDF データを格納することができないという問題点がある．後者はスキーマデータとインスタンスデータを区別なく格納するために，問合せ処理において無駄な検索領域を検索する必要がある．

*奈良先端科学技術大学院大学 情報科学研究科 情報生命科学専攻 博士論文，NAIST-IS-DD0261026, 2005 年 2 月 3 日.

本論文では，上記の問題を解決するために， RDF データを関係データベースへ格納する手法と，経路式の接尾辞配列に基づく RDF データのための索引手法を提案する．提案した関係スキーマは， RDF スキーマデータに依存しない設計で，スキーマとインスタンスを区別して格納することを基本方針とした．手法としては，RDF データを部分グラフへ分割し，それぞれを異なる関係表へ，部分グラフの特性に従って異なったアプローチで格納する．すなわち，スキーマデータを含む部分グラフは，任意の 2 要素間の継承の有無を知ることが重要になるため，インターバルナンバリングスキームを採用し，インスタンスデータを含む部分グラフは，経路式に基づく問合せ処理における結合演算の回数を減少させるために経路式に基づく格納方法を採用する．

RDF データのための索引手法は，RDF データから抽出した部分グラフごとに経路式を生成し，その経路式を文字列に見立てて，接尾辞配列を作成する手法である．与えられた経路式を発見する場合，接尾辞配列を二分探索によって一致する経路式を検索する．この手法は，従来の RDF データベースの索引として利用することができ，経路式に基づいた問合せ処理の結合演算を減少させることが可能になる．

本論文では，実験を通して提案手法の性能評価を行い，大規模な RDF データを効率的に格納できることを確認した．従来の RDF データベースは，経路式が長くなるにつれ，処理時間が増加したのに対し，提案した RDF データベースでは，徐々に処理速度が減少した．これは，経路式が長くなるにつれ，従来の手法では結合演算の回数が増加するために，処理コストが増加したが，提案手法では解の集合が小さくなるために，処理コストが減少したためである．また，接尾辞配列に基づく索引手法の実験では，従来の手法での処理時間は経路式の長さと対象データのサイズの両方に依存していたのに対し，提案手法では，データのサイズにのみ依存し，経路式の長さに依存していないことを確認した．

キーワード

RDF データベース，経路式，有向グラフ，接尾辞配列，ナンバリングスキーム

Studies on Storage and Retrieval for RDF Data Based on Path Exporessions*

Akiyoshi Matono

Abstract

In this study, we propose schemes for efficient storage and retrieval of Resource Description Framework (RDF) data based on path expressions. RDF is a foundation to describe metadata, and is recommended for the fruition of Semantic Web which has emerged as a vision of the next generation of the Web. RDF data is composed of a set of statements, which can be represent as binary relationships among Web resources. The structure of RDF data thus is a directed graph.

Today, it is becoming increasingly common to use RDF as a metadata format. One typical usage is to describe large-scale metadata. In the near future, large RDF-based metadata is considered to increase rapidly as RDF comes into wide spread use. In order to handle such data efficiently, RDF databases and indexing scheme that can manage massive RDF data are essential.

So far, several RDF databases have been proposed. The first problem is poor performance in processing path queries. The reason is that RDF data are decomposed into statements in most of the conventional RDF databases. Therefore, we need to perform a join operation per each path step. This results in performance degradation as query length becomes longer. The second problem concerns the ability to handle RDF schema. The conventional RDF databases can be classified into two groups: the first group is designed depending on RDF schema; and the second group stores RDF data

*Doctoral Dissertation, Department of Bioinformatics and Genomics, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0261026, February 3, 2005.

in terms of statements. The former cannot handle a schemaless RDF data. The latter have to search the superfluous area, where both data are stored, because they do not make any distinction between schema and instance data.

In this study, we propose a scheme to store RDF data in relational databases and an indexing scheme for RDF data based on suffix array of path expressions. Our proposed relational schema is designed to be independent of RDF schematic information, and designed to make the distinction between schema and instance data. In our approach, we first divide RDF data into subgraphs. We then store the subgraphs into distinct relational tables by applying appropriate techniques for representing each subgraph. In particular, we store all reachable path expressions in a subgraph including instance data. In addition, we apply an interval numbering scheme to subgraphs including schematic information, enabling us to efficiently detect ancestor-descendant relationships between two nodes.

Meanwhile, in our indexing scheme based on suffix array, we generate a set of path expressions from subgraphs which is extracted from RDF data. We then construct suffix array from path expressions. When processing path query, we perform binary search of the suffix array. This indexing scheme can be used with the conventional RDF databases and make the number of join operations to be decreasing.

Through a series of experiments to evaluate the performance, we affirm that our approach can efficiently handle massive RDF data. Interestingly, as the path lengths grow, the processing times of the conventional approach increase, while the times of our approach decrease. The reason is that, as the path lengths grow, the number of join operations is increasing, in contrast the size of answer set is decreasing. In experiment of the proposed indexing scheme, the processing times using the conventional scheme depend on path lengths and data size whereas the processing times using our scheme depend on only data size, but not path lengths.

Keywords:

RDF database, path expressions, directed graph, suffix array, numbering scheme

目 次

図 目 次

図 目 次

表 目 次

第1章

はじめに

1.1 研究の背景

インターネットやイントラネットの普及に伴って，利用者は様々な情報に容易に触れることができるようになった．コンピュータネットワークを利用した代表的なアプリケーションとして， World Wide Web (WWW) がある． WWW は主に Hyper Text Markup Language(HTML) 形式のデータを交換することで成り立っている．HTML は，文書の論理構造や見栄えを表現することができ，マルチメディアデータ (静止画，動画，音声など) や，他の文書の位置（ハイパーリンク）を埋め込むことができる．WWW では，静的な HTML 文書を交換するだけでなく，Common Gateway Interface(CGI) や Servlet など，動的なコンテンツの交換も実現している．

WWW の爆発的な普及によって多様で膨大な情報がインターネット上に氾濫している． WWW 上で流通しているコンテンツは，人のために作成されており，計算機のためのコンテンツはほとんど流通していない．言い換えれば， 現在の WWW 上の情報の多くは，人にしか理解することができず，計算機にとっては理解不能である．たとえば，ピカソが描いたゲルニカという作品の画像を埋め込んだ HTML 文書があるとする．人はその HTML 文書をブラウザによって閲覧し，文書内に記述された文章を読むことで，その画像の作者がピカソで，タイトルがゲルニカであることを理解することができる．しかしながら，計算機にとっては，その画像を説明した文章は単なる文字列であるため，画像近傍の文字列がその画像に関連する文字列であると認識することはできたとしても，作者がピカソでタ

イトルがゲルニカであるといった情報を明確に認識することは難しい.

このような問題を解決するために提案されたのが Smenatic Web (SW)[40] である． Semantic Web は， WWW の創始者でもある Tim Berners-Lee により提唱された次世代の Web のビジョンで，現状の WWW 上のコンテンツに計算機が理解可能な意味情報を付加することにより， WWW の有用性を飛躍的に高めようとしている． Semantic Web が実現することで，これまで人手によって行われてきた作業の一部を計算機に任せることができ，利用者の負担を軽減することができるようになる.

Semantic Web では，WWW にはない知的な応用を提供することができる．一つの例として，知的検索処理がある．たとえば,「日曜日に営業している奈良にある歯科」を検索したいとする．この場合，従来の WWW 上で検索した場合，もちろん目的の情報も検索結果として提示されるが,「日曜は休日の奈良橋歯科という名の歯科」も検索結果として提示されることになる．これが Semantic Web 上の場合，利用者が目的としているウェブページには，営業日の中に「日曜日」を含んでおり，場所は「奈良」にあり，職業が「歯科」といった情報が詳細に記述されている．そのため，利用者は，そういった情報を検索のキーとすることで，目的のウェブページのみを検索することができる.

Semantic Web における計算機が理解可能な意味情報とは，メタデータのことを指す．メタデータとは，データのためのデータのことで，あるコンテンツが持つ情報自体ではなく，そのコンテンツに関する情報である．たとえば，静止画の作成日時や製作者，サイズなどが，メタデータにあたる．すなわち，従来の WWW と Semantic Web とのもっとも大きな違いはメタデータであると言える.

Semantic Web を実現するためには，メタデータが大量に存在する必要がある．しかしながら，単にメタデータが大量に存在すればよいわけではなく，Semantic Web のビジョンに基づく原則に沿ったメタデータが豊富に存在する必要がある．その Semantic Web を実現するための原則 (Principles) とは以下の五つである [20].

- 全てのものが識別可能 (Everything Identifiable is on SW)
  Uniform Resource Identifiers (URI) で識別されるものはすべて Semantic Web 上で扱うことができる． Web 上のコンテンツはもちろん，人間や事象，概

念なども識別することができれば，Semantic Web の上でひとつの資源 (Resource) として扱うことができる．

- 部分的な情報でも成立する (Partial Information)
  Semantic Web では完全な情報を目指さない．現在の WWW でもリンクの完全性を保障しないように，Semantic Web でもデータの完全性，一貫性を求めない．すなわち，誰が何についてどのようなことを記述することも自由である．

- 信頼できるウェブ (Web of Trust)
  Semantic Web 上のすべての資源が真実であるとは限らない．そのため，アプリケーションは，その資源の文脈 (Context) を読み取って，述べられていることが真実 (より現実的に言うならば “信頼に値する (trustworthiness)” こと) であるかどうか評価しなければならない．

- 発展性 (Evolution)
  Semantic Web では，様々なところで独自に知識が記述される． Semantic Web に発展性を持たせるためには，多様なコミュニティのそれぞれの試みを効率的に組み合わせるられるようにしなければならない．また，過去の情報を修正することなく，新しい情報を付け加えることができるべきである．

- 最小の制約の設計 (Minimalist Design)
  単純なことは単純のままで，複雑なことは処理可能になることが重要である．できるだけ制約を課さず，必要以上の標準化を求めず，将来複雑なことを実現可能にするために単純なところからはじめる．

Semantic Web におけるメタデータは，これらの原則に基づいて記述される必要がある．そのためメタデータ記述のための仕様は，これらの原則に沿って制定されている．仕様の制定は，Semantic Web 原則の 「最小の制約の設計」 を満たすために，単一の仕様だけ実現するのではなく，複数の仕様を段階的に重ね合わせて行われている．図 1.1 は，Semantic Web のメタデータに関する仕様を階層的に表したレイヤーケーキと呼ばれる図である．その図に，前述した歯医者の例を用いて，各層で記述するメタデータを具体的に示した．

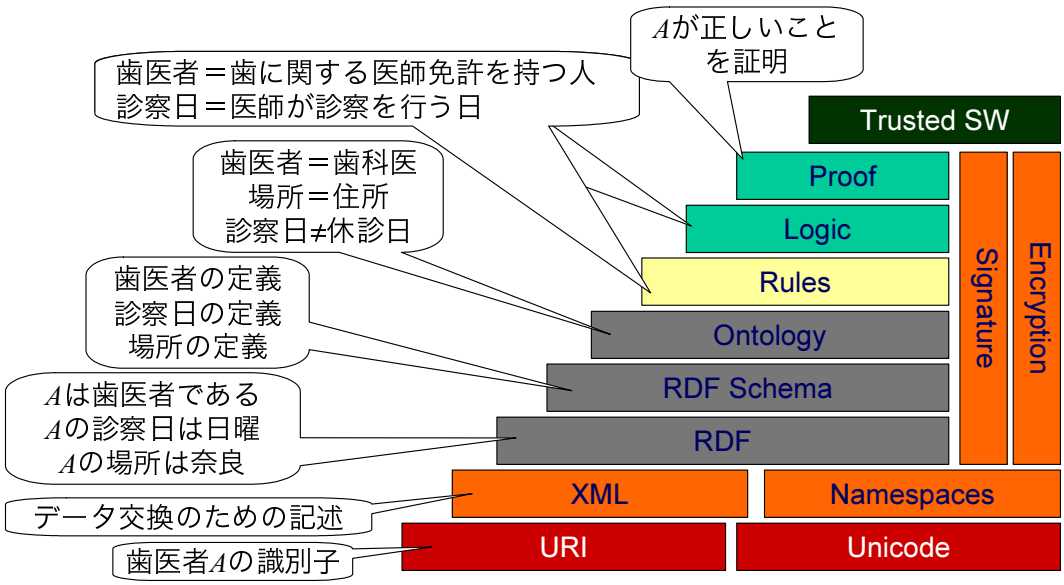

図 1.1 Semantic Web のレイヤーケーキ

レイヤーケーキの各層の役割を簡単に述べる．各層は，それよりも下の層の仕様や技術を利用して実現する．

- **Trust 層**は文脈，証明，暗号化と電子署名により，エージェントが示した結果の信頼性を判断する層．
- **Proof 層**はエージェントの処理の履歴，処理理由など，結果を導いた根拠を証明するための仕様を提供する層．
- **Logic 層**は一階述語論理などを用いた知識の記述と，それに基づくエージェントの処理のための仕様を提供する層．
- **Rule 層**は問い合わせ，フィルタリングを可能にする共通基盤としての論理の定義するための仕様を提供するための層．
- **Ontology 層**はより精密な語彙の定義と，複数のスキーマの関係づけ・融合を可能にする推論を記述ための仕様を提供する層．

- **RDF Schema 層**は語彙 (クラス，プロパティ) を定義することのできる仕様を提供する層.
- **RDF 層**は計算機で処理可能なメタデータを表現するための仕様を提供する層.
- **XML/Namespace 層**は処理が容易な自己記述可能な言語の仕様と複数語彙の区別・混在を可能にするための仕様を提供する層.
- **URI/Unicode 層**は資源を識別するための仕様と多国語処理のための文字コードを提供する層.

レイヤーケーキで示した階層の中で，下二層の URI/Unicode 層と XML/Namespaces 層を担う仕様は，メタデータのための仕様ではなく，一般的なデータを記述するために制定された仕様である．RDF 層以上の層は，メタデータを記述するための仕様である．この RDF 層を担う仕様として，Resource Description Framework (RDF) [47] がある[1]．すなわち， RDF はメタデータ記述のための基礎を定義した仕様で，Semantic Web 実現のための根幹となっている．本論文では，RDF に注目する．

RDF は，文と呼ばれる最小単位で構成されている．文は，主語，述語，および目的語の三つから構成されており，一つの文を用いて，一つの事象を記述することができる．すなわち，主語は述べたい事象の主題を表し，述語によってその主題に関するある特性を特定し，目的語によってその主題におけるその特性の値を表す．具体的には，主語を歯医者，述語を診察日，目的語を日曜日とすることで，その歯医者は日曜日に診察することを表現することができる．文を複数用いることで，RDF では複雑なメタデータを記述することができ，RDF を用いて記述したデータの構造は，有向グラフ構造となる．

従来の WWW 上にもこれまで，メタデータは存在した．たとえば， HTML の META タグや，画像データの Exchangeable Image File Format (EXIF) [2] などがある．

[1] Semantic Web を実現ためのメタデータ記述言語には， XML Topic Map (XTM) [23] も挙げられるが，本論文では， RDF を研究対象としているため， XTM に関する詳細は割愛する．

[2] http://www.exif.org/Exif2-2.PDF

しかしながら，従来のメタデータにはいくつかの問題が残されており， Semantic Web の原則を十分に満たすことができない．従来のメタデータの問題点を以下に列挙する．

1. メタデータ記述のための統一した規格や言語が制定されていないために，異なる仕様のデータには異なる仕様のメタデータ記述が用いられている．そのため，メタデータ間に相互互換性が無い．
2. 多くのメタデータ記述の仕様では，定義されたプロパティが固定で，変更や追加ができないため，利用者による新しいプロパティの定義ができない．
3. あらゆる資源に対してメタデータを付与することができない．
4. 資源同士の関連を表現したメタデータを記述することができない．
5. メタデータの構造が単純で，簡単な内容の情報しか表現できない．

RDF によって上記の問題は以下のように解決される．

1. RDF という統一した規格が存在するために，異なる仕様のデータであっても，すべて RDF に基づいて記述することができる．
2. RDF では，利用者によってプロパティを自由に追加できる．また，ほかにも利用者は多くの定義を行うことができる．
3. RDF では， URI によって識別できるものであれば，資源としてメタデータを表現できる．そのため，対象となる資源はインターネット上のコンテンツはもちろんのこと，インターネット上に存在しないものも対象となる．
4. 単一の RDF 内に複数の資源に関するメタデータを記述することができ，資源間の関連情報も容易に表現できる．
5. RDF の構造は，有向グラフ構造であるため，複雑な情報を構造的に表現できる．

RDF を用いることで，データの意味を表現することができるようになる．これは従来から研究されたきた人工知能分野の論理型言語によっても表現することができる．これらの論理型言語と RDF とのもっとも大きな違いは階層化した点であると考えている．図 1.1 で示したように RDF は Semantic Web のための一部の仕様にしか過ぎず，従来の論理型言語では，図 1.1 のすべての層を一つの言語で補っている．これによって，複雑なルールやロジックの記述は専門家に任せておいて，利用者に密接にかかわる単純な事象のみを記述することができる．また，他の相違点としては，RDF は利用者によって記述されるために，矛盾が発生することを許可している点や，完全な推論や演繹を提供しない点などがあげられる．

1.2 研究の動機

現在， RDF の枠組みを利用して設計されたメタデータ記述フォーマットが多く提案されている．このような RDF に基づく各フォーマットは，メタデータの内容ごとに，異なった仕様として定義される．たとえば，Web サイトの概要を記述するための RDF Site Summary (RSS 1.0) [32] や人々に関する情報とそのつながりを公開，共有するための Friend of a Friend (FOAF) [11]，あらゆる資源が共通して持つ情報を表現するための Dublin Core [14] などは，Weblog などで利用されており，現在爆発的に普及している．

一方， RDF の枠組みに基づいてそれぞれの個別分野の情報を表現するプロジェクトも多く出現してきた．たとえば，オンライン辞書の WordNet [5] ，遺伝子産物の機能に関して注釈付けした Gene Ontology [15] ，世界最大のウェブディレクトリの Open Directory Project [28] などは配布フォーマットとして， RDF による提供も行っている．こういった RDF を用いて個別分野の情報を表現するプロジェクトは，RDF の普及に伴って今後も増加することが予想される．

以上のような背景から今後 RDF によって記述されたデータが増大することは容易に予想できる． RDF で記述されたデータが大量に存在するために，これらを効率的に格納して検索することが重要になる．そこで，大量の RDF データを扱うことができるデータベースや索引手法が重要になる．

RDF によって記述されたメタデータは，構造情報を持っており，構造化文書として扱うことができる．従来から構造化文書に対する索引やデータベースは多く提案されている．しかしながら，従来の構造化文書のためのデータベースや索引は固定したスキーマにのみ対応するよう設計されている．たとえば，新聞記事や科学技術論文などの管理システムである．しかしながら， RDF は，メタデータとして利用することを想定しているために，従来の構造化文書とは異なり，スキーマが厳格に定められるとは限らない．そのため，スキーマを持たない RDF や スキーマ が異なる異種データに対して，統一的な検索が重要である．

RDF が構造を持っており，利用者が自由に構造を決定できる点では，Extensible Markup Language (XML) [41] に非常に類似している．また，RDF データが表すグラフモデルを XML で表現するための構文も定義されている．そのため， RDF データを効率的に扱う手法として， RDF データを XML 文法で記述し，それらを XML データベースや XML のための索引手法などを適用することも考えられる．しかしながら，この手法には大きな問題がある．それは， RDF データを XML 文法で記述した場合，同じ意味を表す複数の表現が存在することである．すなわち， XML の意味モデルの構造は木構造をしているが， RDF グラフの構造は有向グラフ構造であるために， RDF データを XML データベースに格納することは，XML 文法に基づいて格納することになり， RDF グラフに基づいて格納されないことになる．

これまで，RDF データを格納するためのデータベースは幾つか提案されている [2], [3], [6], [36], [25], [31], [33]．しかしながら，これらには，次の二つの問題点がある．一つ目の問題点は，スキーマ情報の扱いに関する問題である．スキーマ情報を利用した問合せには，たとえば “Artist クラスを継承しているすべての子孫クラスを列挙せよ” があり，重要な問合せの一種である．従来の RDF データベースには，スキーマ情報を扱うことを前提にして設計されたもの [2], [6] と，そうでないものとがある [3], [36], [31], [33]．前者は，スキーマ情報を扱うことに関しては優れているが，スキーマ情報を持たない RDF データを格納することができない．一方，後者はスキーマ情報に関する一部の問合せが困難であることが挙げられる．さらに，スキーマ情報とインスタンス情報を区別なく格納するために，問

合せにおいて無駄な検索領域を検索する必要がある．

二つ目は，結合演算の回数が増加し，大規模な RDF データにおける検索速度が低下してしまう点である．その理由について説明する．従来の RDF データベースのほとんどは，関係データベースか Berkley DB [3] を利用して，RDF データを最小単位の文に分割して格納する手法を採用している．そのため，文に基づく問合せは効率的に処理することができる．しかしながら，RDF データ の構造は有向グラフ構造であるため，部分グラフを発見するような問合せや，特定の経路を巡回するような問合せが頻出する．このような問合せを処理する場合，一般的に経路式による検索が行われる．従来の RDF データベースでは，問合せの構造単位と RDF データの格納の構造単位が異なるために，解を得る処理において経路長に応じた結合演算が必要になり，検索性能の低下につながる．

文に基づく問合せとは，主語，述語，目的語のうち，一つあるいは二つが不明である場合，その不明な要素を特定する問合せである．すなわち，文に基づく問合せ処理では，部分的に不明な文を問合せのキーとして用い，不明な要素を補完する処理を行うことで，不明であった部分を解として返す．具体的には，主語が「歯医者」で，述語が「場所」であるような文の目的語が不明であるとする．その二つを問合せとして発行し，解として，目的語「奈良」を得る．次に，経路に基づく問合せについて説明する．まず，経路とは，複数の文が直列に接続されたものである．すなわち，あるの文の目的語とほかの文の主語が一致する場合，それらの文が接続したものを経路という．逆に言えば，RDF によって記述されたメタデータの構造は有向グラフ構造であるため，ある頂点から他の頂点までの道を経路という．経路式に基づく問合せ処理とは，ある経路の一部が不明であるとき，その不明な部分を補完するような処理である．すなわち，文に基づく問合せを直列に接続するよう拡張した問合せである．そのため，文に基づく問合せを複数回繰り返し，解を結合することで，経路式に基づく問合せを処理することはできる．しかし，一般的に結合を行う演算処理はコストが高く，処理時間がかかるという問題がある．

[3] http://www.sleepycat.com/

1.3 研究の概要

従来の RDF データベースの多くは，有向グラフ構造を表現するために，RDF データを最小単位の文に分割し，それぞれを個別に格納する手法を採用している．そのため，文に基づく問合せ処理には効率的であるが，経路に基づく問合せに対しては，効率的な処理は期待できない．そこで本論文では，RDF データを効率的に検索するための手法として，経路式に基づく手法を提案する．これによって経路式に基づく問合せを効率的に処理することができる．本論文では，以下の特徴を持つ二つの手法を提案する．

- 経路式に基づく RDF データベース [52]
  関係データベースを用いた，スキーマデータに依存しない 関係スキーマを設計することを基本方針とした手法である．この手法の特徴は，インスタンスデータを経路式に基づいて格納すことで，結合演算の回数を減少させ，経路式に基づく問合せを効率的に処理できることである．また，スキーマデータとインスタンスデータを区別することで，スキーマデータを持たない RDF データを格納することができ，かつ，スキーマデータを持つ RDF データに対しては，無駄な領域を探索する必要がなくなる．

- 経路式に基づく RDF データのための接尾辞配列を用いた索引手法 [51]
  RDF データから抽出した経路式に対して，経路式の各要素を文字とみなして，接尾辞配列を生成する手法である．特徴として，経路式 RDF データベースでは処理コストが高い，頂点を含む経路式に基づく問合せを効率的に扱うことが出来る．また，従来の RDF データベースで，索引として用いることができる．

本論文では，上記の二つの手法の性能を評価した．経路式に基づく RDF データベースでは，スキーマデータに基づく問合せと経路式に基づく問合せの 2 種類の評価を行った．スキーマデータに基づく問合せは，一般的なスキーマに関する問合せに対して，従来の RDF データベースと比較して，どのような処理を行うかを調査した．提案する手法では，従来の手法に比べ，無駄な領域にアクセスすることなく問合せを処理することができることが確認できた．一方，経路式に基

づく問合せの性能を実験によって評価した結果，経路式の長さが長くなるにつれ，従来の手法では性能が低下したのに対し，提案手法では，解の集合が小さくなるために，徐々に処理速度が向上することを確認した．また，経路式の接尾辞配列に基づく索引手法の性能評価では，従来の手法は処理速度が対象のデータサイズと経路式の長さの両方に依存していたのに対し，提案手法では，データのサイズにのみ依存しており，経路式の長さには依存しないことを確認した．

1.4 本論文の構成

本論文の構成を述べる．まず 第 2 章 では，本論文で扱う RDF とその関連仕様に関して述べる．加えて，現在普及している RDF データに関してそれらの特徴に基づいて大まかに分類する．第 3 章 では，従来提案されている RDF データベースの紹介とそれらの問題点を挙げ，本論文の立場を述べる．第 4 章 では，従来のデータベースにおける問題点を解決するために RDF データの関係データベースへの格納方法を提案し，性能評価を実験を通して行う．また，第 5 章 では，経路式に基づく接尾辞配列を提案し，性能を実験を通して評価する．これによって，従来の RDF データベースを利用したまま，問題点を解決することができる．最後に，第 6 章で，今後の課題として，閉路を含む RDF データに対応する方針を述べ，本研究を通して得た知見，今後の研究の方向性について述べる．

第2章

Resource Description Framework (RDF)

本章では，インターネット上のあらゆる資源に対してメタデータを統一的に表現するための標準規格として制定された Resource Description Framework (RDF) [47] の概要に関して述べる．さらに RDF のスキーマ言語やその他の周辺仕様についても簡単に述べる．

W3C の RDF コアワーキンググループ は，RDF を 1999 年に一度制定したが，その後，不足部分を補足し曖昧な部分を明確にして， 2004 年 2 月 に以下の六つの仕様を同時に制定した．

- RDF/XML Syntax Specification (Revised)[46]
  RDF の XML 構文記述の仕様書.
- RDF Vocabulary Description Language 1.0: RDF Schema [45]
  RDF のスキーマ定義のための仕様.
- RDF Primer [42]
  RDF 関連仕様の要点をまとめている．
- Resource Description Framework (RDF): Concepts and Abstract Syntax [48]
  抽象グラフ構文の定義，設計方針，RDF 文書の意味， キー概念等に関して言及している．

- RDF Semantics[43]
  RDF モデルの意味論に関して議論している.

- RDF Test Cases [44]
  問題点等をテストした結果をまとめたもの.

2.1 RDF の概要

RDF とは，資源に関する情報を表現するための枠組みで，特に，表題，著者，ウェブページの更新日，著作権やライセンス情報，共有資源の有効スケジュールなどのようなウェブ資源に関してのメタデータを表現することを目的として設計されたものである.

2.1.1 RDF の基本概念

RDF は次に示す重要な概念を持っている.

グラフデータモデル

RDF に基づいて表される情報は，主語 ，述語 ，および目的語 から成る文 (トリプルとも呼ばれる) の集まりである.

- 主語 (subject) は，URI 参照，あるいは空白ノードで表される.
- 述語 (predicate) は，URI 参照で表される.
- 目的語 (object) は，URI 参照，あるいはリテラル，空白ノードで表される.

文の集合は， RDF グラフと呼ばれる． RDF グラフは，頂点と頂点とを有向辺によって接続することで，それぞれの文を表すことができる．具体的に，文は図 2.1 に示すように表される．RDF グラフの頂点は文の主語と述語で，有向辺は述語を意味している．有向辺の向きは，常に主語から目的語に向かっている.

図 2.1 RDF の文 (statement)

RDF の文は，主語と目的語が述語の関係で成り立っていることを示している．具体的に，文を用いて 「ウェブページ (http://www.matono.net/index.html) の作者は Akiyoshi MATONO である」という事象を表現する場合，主語と述語，目的語はそれぞれ次のようになる

- 主語は http://www.matono.net/index.html という URI 参照である．
- 述語は作者を意味する URI 参照 creator である．
- 目的語は Akiyoshi MATONO というリテラルである．

RDF グラフに含まれるすべての文が有効であることを示しているため，RDF グラフではすべての文が論理的な AND で接続される．

頂点の識別

頂点は，フラグメント識別子付きの URI (URI 参照， URIref) ，リテラル，あるいは空白ノードのいずれかである．プロパティは URI 参照である．URI 参照とリテラルは，その頂点を表す識別子として用いられる．また，URI 参照は頂点によって表されるもの同士の関係を表す述語の識別子として用いられる．述語のための識別子は RDF グラフ内において頂点としても用いられる．空白ノードは，URI 参照やリテラルではなく，文の中で一回以上用いることができる唯一の頂点である．

データタイプ

データタイプは整数や浮動小数点，日付など値の型として用いられる． RDF では， XML 表現で用いるための `rdf:XMLLiteral` というデータタイプを持っているが，その他の数字や日付などの一般的なデータタイプは定義されていない．RDF では，テータタイプの定義を分離させており，代わりに XML Schema [39] のデータタイプを RDF のために利用することを想定している． RDF では，新しいデータタイプを定義する手段を提供していないが，一方 XML Schema では RDF で用いることのできる新しいデータタイプを定義するための枠組みを提供している．

リテラル

リテラルは数字や日付などの値を識別するために用いられる．リテラルによって表現されたあらゆるものは URI によっても表現されることができるが，より便利で理解しやすくするためにリテラルが用いられる．

リテラルはプレインと型付の二種類あり，プレインリテラルは言語タグを付加できる文字列で，型付リテラルはデータタイプ URI を付加した文字列である．

2.1.2 RDF の意味論

RDF の意味論を示す．URI 参照とリテラルの集合 V に対し， V の解釈は，$I = \langle IR, IP, IEXT, IS, IL, ILV \rangle$ の組で，それぞれは以下のように定義される．

1. IR は資源の非空の集合で， I の議論領域あるいは全体空間と呼ばれる．
2. 集合 IP は I のプロパティの集合で， IR のサブセットである．
3. $IEXT$ は，プロパティに意味を与えるために使用され， IP から $IR \times IR$ の冪集合への写像である．
4. IS は， V に含まれる URI 参照から， IR と IP の和集合への写像である．

5. IL は，V に含まれる型付きリテラルから，IR への写像である．
6. LV は IR のサブセットで，V に含まれるすべてのプレインリテラルの集合である．

RDF グラフの明示的な意味は，以下の規則によって再帰的に与えられる．以下で，= は左右辺が等しいことを意味し，$\langle x, y \rangle$ は x と y の順序を持つ組を表す．さらに，リテラルはダブルクォーテーション (") で囲み，アットマーク (@) によって言語タグを示し，トリプルの末尾にはピリオド (.) をつける．

- もし E が V に含まれるプレインリテラル "aaa" であれば，$I(E) = aaa$ である．
- もし E が V に含まれるプレインリテラル "aaa"@ttt であれば，$I(E) = \langle aaa, ttt \rangle$ である．
- もし E が V に含まれる型付きリテラルであれば，$I(E) = IL(E)$ である．
- もし E が V に含まれる URI 参照であれば，$I(E) = IS(E)$ である．
- もし E がトリプル $s\ p\ o.$ であれば，$s \in V$，$p \in V$，$o \in V$，$I(p) \in IP$，かつ $\langle I(s), I(o) \rangle \in IEXT(I(p))$ を満たすとき，$I(E) = true$ である．そうでなければ，$I(E) = false$ である．
- もし E が RDF グラフであれば，E に含まれる幾つかの E' において $I(E') = false$ を満たすとき，$I(E) = false$ である．そうでなければ，$I(E) = true$ である．

2.1.3 RDF 語彙の定義

RDF は仕様内で定義された幾つかの語彙を持っている．利用者はそれらと，RDF スキーマデータで定義された語彙を利用することで，RDF データを記述する．たとえば，rdf:type を述語，rdf:Property を目的語として用いた文を作成することで，

その主語はプロパティであることを定義することができる．本節ではそれら RDF 名前空間で用いる語彙の定義を示す.

語彙集合 V の解釈は，下記条件を追加した， URI 参照とリテラルの集合 V と RDF 語彙集合 rdfV との和集合の解釈 I とする．これらのトリプルは暗黙的なトリプルと呼ばれる．ここで，ある文字列 sss が rdf:XMLLiteral の語彙空間に属するのであれば， XML リテラル型文字列 (well-typed XML literal string) であるといい． XML リテラル型文字列に対応する値をそのリテラルの XML 値 (XML value) と呼ぶ．また，^^ はリテラルの型を表す.

- $x \in IP \Leftrightarrow \langle x, I(\texttt{rdf:Property})\rangle \in IEXT(I(\texttt{rdf:type}))$
- もし "*xxx*"^^rdf:XMLLiteral $\in V$ かつ *xxx* が XML リテラル型文字列であるならば,
  IL("*xxx*"^^rdf:XMLLiteral) は *xxx* の XML 値，かつ
  IL("*xxx*"^^rdf:XMLLiteral) $\in LV$，かつ
  $IEXT(I(\texttt{rdf:type}))$ は $\langle IL(\text{"}xxx\text{"}\texttt{\^{}\^{}rdf:XMLLiteral}), I(\texttt{rdf:XMLLiteral})\rangle$ を含む.
- もし "*xxx*"^^rdf:XMLLiteral $\in V$ かつ *xxx* が XML リテラル型文字列でないならば,
  IL("*xxx*"^^rdf:XMLLiteral) $\notin LV$，かつ
  $IEXT(I(\texttt{rdf:type}))$ は $\langle IL(\text{"}xxx\text{"}\texttt{\^{}\^{}rdf:XMLLiteral}), I(\texttt{rdf:XMLLiteral})\rangle$ を含まない.

2.1.4 RDF 構文

前述したように RDF の基本モデルは有向グラフである． RDF はそのグラフを表現するのための基礎を提供することが目的であるため，RDF グラフを記述する構文は必ずしも一つとは限らない．たとえば，グラフで表現する方法には，Extensible Markup Language (XML) 文法に基づいた構文 (RDF/XML) [46] ，表記を簡略化して扱いやすくした Notation3 (N3) [35]， Notation3 をさらに簡略化した N-Triples [12] がある.

本節では，後述の説明のために単純な構文である Notation3 [35] と，RDF 表現のもっとも代表的な構文である RDF/XML[46] の文法について述べる．

Notation3

RDF データを表現する方法には，グラフ， RDF/XML などがあるが，第 3 の扱いやすい記法として， Notation3 (N3) [35] が提案されている．単純で理解しやすいだけでなく， RDF/XML より表現力が豊かで， Semantic Web レイヤーケーキの Rule や Logic 層の情報を書き表すことが可能である．

N3 文法の BNF による定義の一部を以下に示す.

図 2.2 Notation3 文法文法 (BNF)

```
1  document ::= statementlist period
2  statementlist ::= subject propertylist |
3                  statement period statementlist
4  statement ::= subject propertylist |
5              directive
6  directive ::= '@prefix' prefix ':' uri-ref2 period
7  propertylist ::= prop objectlist |
8                 prop objectlist ';' propertylist
9  objectlist ::= object |
10              object ',' objectlist
11 subject ::= node
12 prop ::= node
13 object ::= node
14 node ::= uri-ref2 |
15         number |
16         string
17 uri-ref2 ::= qname | < URI-Reference >
18 string ::= '"' value '"'
19 period ::= '.'
```

N3 文法の基本は，主語，述語，目的語に示す URI 参照を <> で囲って列挙し，文の最後はピリオド (.) で終わる． 11 行目から 13 行目の subject と prop，object がそれぞれ主語，述語，目的語となる．

N3 には特徴的な構文がある．4 行目の statement が一つの主語から始まる複

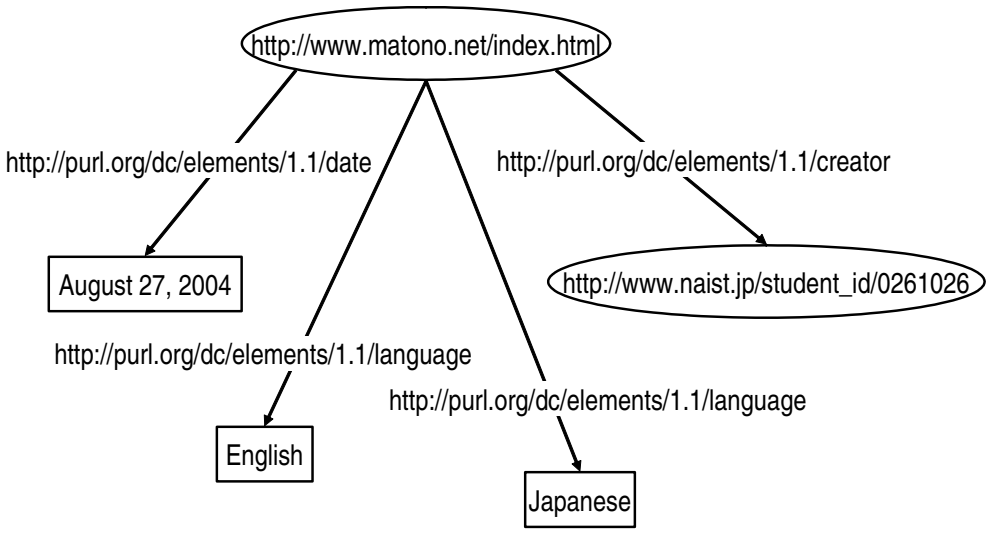

図 2.3 単純な RDF グラフ

数の文を示している．すなわち，主語が一致する複数の文は，主語を省略して述語と目的語を複数列挙することができる．その複数の述語と目的語の列挙を定義しているのが 7 行目の `propertylist` である．この `propertylist` は，共通な述語がある場合，共通な述語を一つ記述し，その直後に目的語を複数列挙することができる．すなわち，共通な主語を持つ文は併合でき，さらに共通な主語と述語を持つ文も併合できる．

N3 は図 1.1 の Rule 層や Logic 層の記述にも用いられることが想定されており，全数限量子や存在限量子の記述方法が用意されているが，本論文では RDF 層と RDF Schema 層を対象としているため割愛する．

N3 を用いて，図 2.3 に示す RDF グラフを記述した例を図 2.4 に示す．

図 2.4 Notation 3 による記述例

```
1  <http://www.matono.net/index.html>
2     <http://purl.org/dc/elements/1.1/creator>
3        <http://www.naist.jp/student_id/0261026> .
4
5  <http://www.matono.net/index.html>
6     <http://purl.org/dc/elements/1.1/date> "August 27, 2004" .
```

```
7
8   <http://www.matono.net/index.html>
9      <http://purl.org/dc/elements/1.1/language> "English" .
10
11  <http://www.matono.net/index.html>
12     <http://purl.org/dc/elements/1.1/language> "Japanese" .
```

1 行目から 3 行目までが一つの文を表している．すなわち，1 行目が主語で，2 行目が述語，3 行目が目的語である． 5–6 行目と 8–9 行目の文では，目的語が資源ではなく，リテラルであるためダブルクォーテーション (") で囲んでいる． また，この例では，主語が一致するため，次のように書き換えることができる．

```
1   <http://www.matono.net/index.html>
2      <http://purl.org/dc/elements/1.1/creator>
3         <http://www.naist.jp/student_id/0261026> ;
4      <http://purl.org/dc/elements/1.1/date> "August 27, 2004" ;
5      <http://purl.org/dc/elements/1.1/language> "English" ;
6      <http://purl.org/dc/elements/1.1/language> "Japanese" .
```

このように，同一の主語を持つ文は，セミコロン (;) を使って主語を略して接続することができる．さらに，主語と述語が一致する二つの文を接続するには，以下のように，カンマ (,) で区切って列挙することができる．

```
1   <http://www.matono.net/index.html>
2      <http://purl.org/dc/elements/1.1/creator>
3         <http://www.naist.jp/student_id/0261026> ;
4      <http://purl.org/dc/elements/1.1/date> "August 27, 2004" ;
5      <http://www.example.org/terms/language> "English" , "Japanese" .
```

一般的な RDF は，名前空間を用いて語彙を区別することができる． RDF/XML では xmlns 属性で名前空間 URI 参照と接頭辞を結びつけ，接頭辞 + ローカル名 (QName) を要素名に用いるが， N3 でも同様に `@prefix` キーワードで QName を利用することが可能である．すなわち，図 2.4 は次のように書き換えることができる．

```
1   @prefix dc: <http://purl.org/dc/elements/1.1/> .
2   <http://www.matono.net/index.html>
3      dc:creator <http://www.naist.jp/student_id/0261026> ;
```

```
4       dc:date "August 27, 2004" ;
5       dc:language "English" , "Japanese".
```

RDF/XML

RDF/XML[46] は，RDF に基づいたメタデータを記述する上で，もっとも一般的に利用されている文法である．RDF/XML は，インターネット上のデータ交換のためのフォーマットとしてデファクトスタンダードとなっている XML を利用した構文である．RDF データを XML 構文で表現するために，XML のためのアプリーケーションを利用することができる．そのため，現在普及している RDF データのほとんどはこの文法に基づいて記述されている．

RDF/XML 文法定義の一部を以下に示す．RDF/XML 文法は XML 文法の範囲内で記述される．より簡単に定義を記述するために，BNF に似た記法によって RDF/XML 文法の定義を行う．

図 2.5 RDF/XML 文法定義

```
1  coreSyntaxTerms ::= rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
2                      rdf:resource | rdf:nodeID | rdf:datatype
3  nodeElementURIs ::= anyURI - ( coreSyntaxTerms )
4  propertyElementURIs ::= anyURI - ( coreSyntaxTerms | rdf:Description )
5  propertyAttributeURIs ::= anyURI - ( coreSyntaxTerms | rdf:Description )
6  doc ::= root(document-element == RDF, children == list(RDF))
7  RDF ::= start-element(URI == rdf:RDF, attributes == set())
8         nodeElementList
9         end-element()
10 nodeElementList ::= (nodeElement)*
11 nodeElement ::= start-element(URI == nodeElementURIs
12                               attributes ==
13                                 set((idAttr | nodeIdAttr | aboutAttr )?,
14                                      propertyAttr*))
15                 propertyEltList
16                 end-element()
17 propertyEltList ::= (propertyElt)*
18 propertyElt ::= resourcePropertyElt | literalPropertyElt | emptyPropertyElt
19 resourcePropertyElt ::= start-element(URI == propertyElementURIs,
```

```
                                    attributes == set(idAttr?))
                      nodeElement
                      end-element()
literalPropertyElt ::= start-element(URI == propertyElementURIs,
                                  attributes == set(idAttr?,
                                                    datatypeAttr?))
                     text()
                     end-element()
emptyPropertyElt ::=
      start-element(URI == propertyElementURIs,
                    attributes == set(idAttr?,
                                      ( resourceAttr | nodeIdAttr )?,
                                      propertyAttr*))
      end-element()
idAttr ::= attribute(URI == rdf:ID, string-value == rdf-id)
nodeIdAttr ::= attribute(URI == rdf:nodeID, string-value == rdf-id)
aboutAttr ::= attribute(URI == rdf:about, string-value == URI-reference)
propertyAttr ::= attribute(URI == propertyAttributeURIs,
                           string-value == anyString)
resourceAttr ::= attribute(URI == rdf:resource, string-value == URI-reference)
datatypeAttr ::= attribute(URI == rdf:datatype, string-value == URI-reference)
```

以下に，RDF/XML 文法の定義に用いた用語の意味を示す．イベントとは，XML Infoset のアイテムに基づいて定義された XML の基本単位である．

- *
  直前の用語を 0 回以上繰り返す．
- ?
  直前の用語が 0 回か 1 回用いられる．
- +
  直前の用語を 1 回以上繰り返す．
- A == B
  イベント要素 A が式 B に一致する．
- A != B
  A が B に一致しない．

- *A* | *B* | ...
  各用語 *A*, *B*, ... から一つ選択.
- *A* - *B*
  A に含まれる用語からすべての *B* に含まれる用語を除去.
- `anyURI.`
  任意の URI.
- `anyString.`
  任意の文字列.
- `list(`*item*1, *item*2, ...`)`
  イベントの順序を持つリスト.
- `set(`*item*1, *item*2, ...`)`
  イベントの集合.
- `root(`*acc*1 `==` *value*1`,` *acc*2 `==` *value*2`,` ...`)`
  Root イベント.
- `start-element(`*acc*1 `==` *value*1`,` *acc*2 `==` *value*2`,` ...`)`
  children `end-element()`
  Element イベント，空要素になりえる子要素 *children*，および End Element イベント.
- `attribute(`*acc*1 `==` *value*1`,` *acc*2 `==` *value*2`,` ...`)`
  Attribute イベント.
- `text()`
  Text イベント.

RDF/XML 文法は非常に自由度が高く，同一の RDF グラフを持つ RDF データであっても様々な記述方法がある．RDF/XML 文法の基本は，単一の文の主語，述語，目的語の関連を示すには，XML 要素の入れ子構造，あるいは属性によって表す．次に，図 2.3 で示した RDF の例を RDF/XML 文法に基づいて記述した例を図 2.6 に示す.

図 2.6 RDF/XML による記述例

```
1  <?xml version="1.0"?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3           xmlns:dc="http://purl.org/dc/elements/1.1/">
4    <rdf:Description rdf:about="http://www.matono.net/index.html">
5      <dc:date>August 27, 2004</dc:date>
6      <dc:language>English</dc:language>
7      <dc:creator
8           rdf:resource="http://www.naist.jp/student_id/0261026"/>
9    </rdf:Description>
10 </rdf:RDF>
```

1 行目の<?xml version="1.0"?>は，XML 宣言であり，以下のコンテンツは XML であるということと，XML のバージョンが何かということを述べている．

2 行目では，rdf:RDF 要素を開始している．ここから始まり，10 行目の</rdf:RDF>で終わる XML 要素は RDF によって記述されたメタデータであることを意味する．同じ行の rdf:RDF の後に続くのは，XML 名前空間宣言で，rdf:RDF 要素の xmlns 属性として表現されている．この宣言では，rdf: が接頭辞についている要素はすべて http://www.w3.org/1999/02/22-rdf-syntax-ns# で識別されている名前空間の一部であるということを明記している．この名前空間は，RDF/XML で使用される RDF 表現のための語彙が定義されている．同様に，3 行目も XML 名前空間宣言で，dc: から始まる要素は，Dublin Core の名前空間 (http://purl.org/dc/elements/1.1/) に属する．

4 行目から 9 行目では，表現している三つの文の RDF/XML を示している．4 行目の rdf:Description 要素の開始タグは，資源の記述 (description) を開始するということを示しており，その文の主語である資源の URI 参照を指定するために rdf:about 属性を使用している．5 行目の要素では，<dc:date> 文の作成日プロパティであることを示しており，その値としてリテラル August 27, 2004 を保持することを示している．このプロパティは，rdf:Description 要素内に入れ子になっており，rdf:Description 要素の rdf:about 属性で指定されている資源<http://www.matono.net/index.html> を主題としている．6 行目も，5 行目と同様に言語プロパティとしてその値に English を保持している．また，この要素も rdf:Description 要素内に入れ子になっているため，5 行目と同様の

資源<http://www.matono.net/index.html> を主語とする．

7 行目と 8 行目の dc:creator 要素は，リテラルではなく別の資源 http://www.naist.jp/student_id/0261026 を目的語として表現している．5 行目や 6 行目の要素のリテラル値を書いたのと同じ方法で，この資源の URI 参照を開始タグと終了タグの間に文字列として書き込んだ場合，dc:creator 要素の値は，URI 参照として解釈されている資源ではなく，文字列 http://www.naist.jp/student_id/0261026 となる． 9 行目は，この特定の rdf:Description 要素の終了を示したタグである．

2.1.5 RDF スキーマ

RDF は，プロパティとその値を使用して，資源に関しての簡単な文を表現する方法を提供する．一方，記述するメタデータの分野によって，特定の語彙 (クラスやプロパティ) のみを使用する状況がある．たとえば，友人関係を表現するメタデータでは，人物や人物間の関係，社会的地位などの語彙が利用される．ほかの例では，絵画を表現するメタデータでは，画風や作品名，サイズ，作者などの語彙が利用される．具体的には，画家 Pablo Picasso を示す資源に関するメタデータを記述する場合，その資源は芸術家 (Artist) クラスの実体として表現され，その資源は絵画 (Painting) クラスの実体である Guernica を描画 (creates) プロパティの値とする．ここで，実体とは，あるクラスに属する資源のことである．

このように，ある資源のメタデータを記述する際，その資源の背景に伴って，特定の語彙，つまりクラスやプロパティを必要とする．しかしながら， RDF 自身はそれらを定義するすべを持たない．そういった語彙を定義するために提案されたのが， RDF Schema [45] である． RDF Schema は，特定のクラスやプロパティを語彙の一つとして定義し，同時に使用されるクラスやプロパティを示すために必要な機構を提供している．たとえば，Artist クラスを記述する際には，creates プロパティが使用されるなどである．すなわち， RDF Schema は， RDF データへの型システムを提供している．なお，この RDF Schema も RDF の枠組みで扱われる．すなわち， RDF Schema を用いて定義するクラスやプロパティなどの語

彙も主語，述語，目的語のトリプルを用いて定義される．

RDF Schema 語彙の定義

RDF Schema では，幾つかの語彙が定義されており，利用者はそれらと RDF 語彙を利用して RDF スキーマデータの記述を行う．本節では，RDF Schema 語彙の定義を行う．

クラスは `rdfs:Class` 型に属するものとして定義され，解釈 I におけるクラスの集合を IC とする．また，$ICEXT$ は IC から IR の部分集合への写像である．

語彙集合 V の解釈は，下記条件を追加した， URI 参照とリテラルの集合 V と RDF 語彙集合 rdfV と RDF Schema 語彙集合 rdfsV の和集合の解釈 I とする．これらのトリプルと RDF Schema の暗黙的なトリプルという．

- $x \in ICEXT(y) \Leftrightarrow \langle x, y\rangle \in IEXT(I(\texttt{rdf:type}))$
  $IC = ICEXT(I(\texttt{rdfs:Class}))$
  $IR = ICEXT(I(\texttt{rdfs:Resource}))$
  $LV = ICEXT(I(\texttt{rdfs:Literal}))$

- $\langle x, y\rangle \in IEXT(I(\texttt{rdfs:domain}))$ かつ $\langle u, v\rangle \in IEXT(x) \Leftrightarrow$
  $u \in ICEXT(y)$

- $\langle x, y\rangle \in IEXT(I(\texttt{rdfs:range}))$ かつ $\langle u, v\rangle \in IEXT(x) \Leftrightarrow$
  $v \in ICEXT(y)$

- $IEXT(I(\texttt{rdfs:subPropertyOf}))$ は IP 上において推移的でかつ反射的である．

- $\langle x, y\rangle \in IEXT(I(\texttt{rdfs:subPropertyOf})) \Leftrightarrow$
  $x \in IP$ かつ $y \in IP$ かつ $IEXT(x) \subset IEXT(y)$

- $x \in IC \Leftrightarrow$
  $\langle x, I(\texttt{rdfs:Resource})\rangle \in IEXT(I(\texttt{rdfs:subClassOf}))$

- $\langle x, y \rangle \in IEXT(I(\texttt{rdfs:subClassOf})) \Leftrightarrow$
  $x \in IC$ かつ $y \in IC$ かつ $ICEXT(x) \subset ICEXT(y)$
- $IEXT(I(\texttt{rdfs:subClassOf}))$ は IC 上において推移的でかつ反射的である．
- $x \in ICEXT(I(\texttt{rfds:ContainerMembershipProperty})) \Leftrightarrow$
  $\langle x, I(\texttt{rdfs:member}) \rangle \in IEXT(I(\texttt{rdfs:subPropertyOf}))$
- $x \in ICEXT(I(\texttt{rdfs:Datatype})) \Leftrightarrow$
  $\langle x, I(\texttt{rdfs:Literal}) \rangle \in IEXT(I(\texttt{rdfs:subClassOf}))$

クラスの定義

RDF Schema の型システムは，オブジェクト指向のそれと類似している．すなわち，ある資源は一つ以上のクラスの実体として定義でき，クラスの継承関係を階層的に表現するすることもできる．たとえば，Artist クラスは，Person クラスのサブクラスで，かつ Painter クラスのスーパークラスであると定義できる．つまり，Painter クラスの実体である資源は，Artist クラスの実体でもあり，Person クラスの実体でもある．

RDF Schema を用いてクラスを定義する場合，定義されるクラスは，値が RDF Schema の仕様内で定義された資源 rdfs:Class であるような rdf:type プロパティをもつ任意の資源となる．したがって，Artist クラスを定義した文は，N3 で表現すると次のようになる．

```
1  ex:Artist rdf:type rdfs:Class .
```

ex: は，Artist クラスの名前空間を表す接頭辞であると仮定している．また，プロパティ rdf:type は，資源がクラスの実体であることを示すために使用される．そのため，画家 Pablo Picasso を表す資源<www.picasso.net> が，ex:Artist クラスの実体であることを表現する文は次のようになる．

```
1  <www.picasso.net> rdf:type ex:Artist .
```

次に，クラス間の継承関係について述べる．ex:Artist クラスを定義するだけではなく，Painter クラスや Sculptor クラスを定義する場合，これらはい

ずれも ex:Artist クラスのサブクラスである．二つのクラスのスーパーセットとサブセットの関係（継承関係）を表すには，RDF Schema の仕様で定義された rdfs:subClassOf プロパティを使用する．具体的には，次のようになる．

```
ex:Artist rdf:type rdfs:Class .
ex:Painter rdf:type rdfs:Class ; rdfs:subClassOf ex:Artist .
ex:Sculptor rdf:type rdfs:Class ; rdfs:subClassOf ex:Artist .
```

プロパティの定義

RDF Schema では，クラスの定義に加え，クラスを特徴づける特定のプロパティを記述できなければならない． RDF Schema では，プロパティは RDF Schema の仕様で定義されているクラス rdf:Property と プロパティrdfs:domain，rdfs:range， rdfs:subPropertyOf を使用して記述される．

RDF のプロパティは，すべて rdf:Property クラスの実体として定義される．そのため， ex:creates などの新しいプロパティは，プロパティに URI 参照を割り当て，値が資源 rdf:Property であるような rdf:type プロパティでその資源を記述することによって表現される．つまり，以下の RDF 文によってそれを表すことができる．

```
ex:creates rdf:type rdf:Property .
```

また， RDF Schema では，プロパティとクラスが RDF データ内で同時にどのように使用されるのかを定義できる必要がある．すなわち，定義されたプロパティが特定のクラスでのみ使用され，特定のクラスのみを値とすることを定義できる必要がある．具体的には，ex:creates プロパティを述語とする場合，ex:Atrist クラスの実体である資源を主語とし，ex:Artifact クラスの実体資源を目的語とする．これらを表現するには， RDF Schema の仕様で定義されているプロパティ rdfs:range と rdfs:domain を使用した文を記述する．

rdfs:range プロパティは，特定のプロパティの値が指定したクラスの実体であるということを述べるために使用される．たとえば，ex:creates プロパティは，ex:Artifact クラスの実体を値とすることを定義したい場合，以下の文を用

いる．

```
ex:Artifact rdf:type rdfs:Class .
ex:creates rdf:type rdf:Property ; rdfs:range ex:Artifact .
```

上記の文では， ex:Artifact がクラスで， ex:creates がプロパティであり，ex:creates を使用している文では，目的語として，ex:Artifact の実体が含まれているということを示している．この ex:Artifact クラスのことを ex:creates プロパティの値域クラスという．

目的語には，資源だけではなく，リテラルを用いることが出来るため，rdfs:range プロパティは値として型付きリテラルであることを示すこともできる．たとえば，プロパティ ex:title には XML データタイプ xsd:string の値があるということを述べる場合，以下の RDF 文を記述する．

```
ex:title rdf:type rdf:Property ; rdfs:range xsd:string .
```

rdfs:domain プロパティは，あるプロパティが用いられる資源のクラスを特定するために使用される．たとえば，ex:creates プロパティは ex:Artist クラスの実体で用いられると定義する場合，以下の RDF 文を用いる．

```
ex:Artist rdf:type rdfs:Class .
ex:creates rdf:type rdf:Property ; rdfs:domain ex:Artist .
```

上記の文は， ex:Artsit がクラスで， ex:creates がプロパティであり， ex:creates プロパティを用いた文では，主語として ex:Artist の実体を含むことを示している．この ex:Artist クラスのことを ex:creates プロパティの定義域クラスという．

加えて，RDF Schema では，クラスと同様にプロパティにおいてもスーパーセットとサブセットの関係（継承関係）を表すことができる．そのためには，rdfs:subPropertyOf プロパティを用いた文を記述する．たとえば，ex:paints プロパティと ex:sculpts プロパティはいずれも ex:creates プロパティのサブプロパティであることを表す文は次のようになる．

```
ex:creates rdf:type rdf:Property .
ex:paints rdf:type rdf:Property ; rdfs:subPropertyOf ex:creates .
ex:sculpts rdf:type rdf:Property ; rdfs:subPropertyOf ex:creates .
```

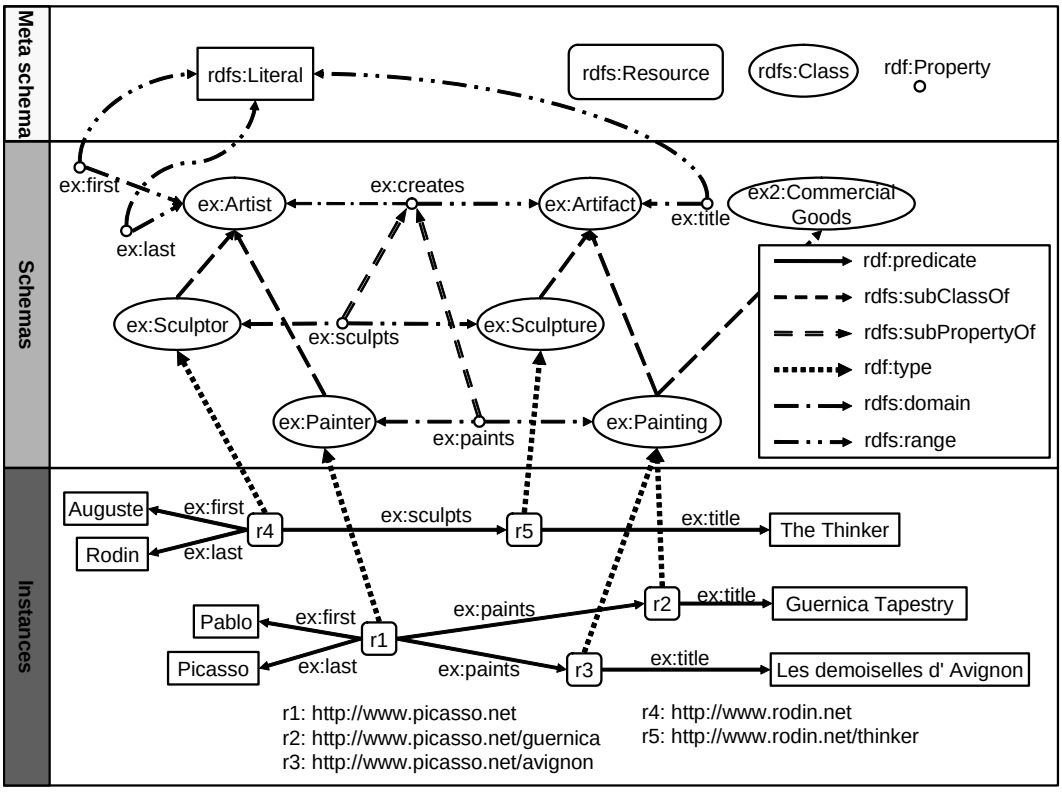

図 2.7 芸術家に関する RDF グラフ

2.1.6 RDF データの例

本節では，これまで述べてきた RDF と RDF Schema を用いた一つの例を挙げる．図 2.7 は，芸術家に関する資源に対してメタデータを付与した RDF グラフを表した例である．

図の上部は， RDF Schema の仕様中で定義された語彙を示しており，スキーマ記述のためのスキーマである．言い換えれば，メタスキーマデータである[1]．中央部分は， RDF Schema を用いて定義された RDF スキーマデータを示している． これまで述べてきた例を組み合わせた例で，芸術家に関するクラスとプロパ

[1]この図は，紙面の制限上，省略してある．正確には記述した以外にも定義されたメタクラスやメタプロパティはいくつかある．また，簡単のために述語を表す矢印も省略している部分がある．

ティを定義し，さらにクラス間の継承関係，プロパティ間の継承関係，プロパティの定義域，値域を定義している．ここで，ex:Artist クラスには，ex:first プロパティと ex:last プロパティがあり，姓と名を記述することができる．一方，ex:Artsit クラスのサブクラスである ex:Sculptor クラスと ex:Painter クラスには，それらのプロパティが無い．しかしながら，クラス間の継承関係によって，サブクラスは，スーパークラスのプロパティを継承するため，ex:Sculptor クラスも ex:Painter クラスも暗黙的に ex:first プロパティと ex:last プロパティの定義域クラスであることを示している．図の下部は，芸術家に関する資源のメタデータを RDF スキーマデータを用いて表現した RDF グラフである．すなわち，これらは， RDF スキーマデータのインスタンスである．本論文ではこれらを RDF インスタンスデータと呼ぶ.

図 2.7 で示した RDF データのスキーマデータ部分 (中央部) を N3 を用いて表すと図 2.8 のようになる.

図 2.8 芸術家に関する RDF スキーマデータ (N3)

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://www.matono.net/example> .
@prefix ex2: <http://www.matono.net/example2> .
ex:Artifact rdf:type rdfs:Class .
ex:Artist rdf:type rdfs:Class .
ex:Painter rdf:type rdfs:Class ;
           rdfs:subClassOf ex:Artist .
ex:Painting rdf:type rdfs:Class ;
            rdfs:subClassOf ex:Artifact ,
                            ex2:CommercialGoods .
ex:Sculptor rdf:type rdfs:Class ;
            rdfs:subClassOf ex:Artist .
ex:Sculpture rdf:type rdfs:Class ;
             rdfs:subClassOf ex:Artifact .
ex:creates rdf:type rdf:Property ;
           rdfs:domain ex:Artist ;
           rdfs:range ex:Artifact .
ex:paints rdf:type rdf:Property ;
          rdfs:domain ex:Paints ;
```

```
          rdfs:range ex:Painting ;
          rdfs:subPropertyOf ex:creates .
ex:sculpts rdf:type rdf:Property ;
           rdfs:domain ex:Sculptor;
           rdfs:range ex:Sculpture ;
           rdfs:subPropertyOf ex:creates .
ex:title rdf:type rdf:Property ;
         rdfs:domain ex:Artifact ;
         rdfs:range rdfs:Literal .
ex:first rdf:type rdf:Property ;
         rdfs:domain ex:Artist ;
         rdfs:range rdfs:Literal .
ex:last rdf:type rdf:Property ;
        rdfs:domain ex:Artist ;
        rdfs:range rdfs:Literal .
ex2:CommercialGoods rdf:type rdfs:Class .
```

一方，図 2.7 で示した RDF データのインスタンスデータ部分 (下部) を N3 を用いて表すと図 2.9 のようになる.

図 2.9 芸術家に関する RDF インスタンスデータ (N3)

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: <http://www.matono.net/example> .
<http://www.picasso.net> rdf:type ex:Painter ;
                         ex:first "Pablo" ;
                         ex:last "Picasso" ;
                         ex:paints <http://www.picasso.net/guernica> ,
                                   <http://www.picasso.net/avignon> .
<http://www.picasso.net/guernica> rdf:type ex:Painting ;
                                  ex:title "Guernica Tapestry" .
<http://www.picasso.net/avignon> rdf:type ex:Painting ;
                                 ex:title "Les demoiselles d' Avignon" .
<http://www.rodin.net> rdf:type ex:Sculptor ;
                       ex:first "Auguste" ;
                       ex:last "Rodin" ;
                       ex:sculpts <http://www.rodin.net/thinker> .
<http://www.rodin.net/thinker> rdf:type ex:Sculpture ;
                               ex:title "The Thinker" .
```

2.2 RDF データの特徴

本節では，インターネット上に普及している各種の RDF データの特徴に関して考察する．各種の RDF データはメタデータとして記述する資源によって使用用途やスキーマデータ，作成者などが大きく異なっている．実際に普及している RDF データを具体的に挙げ，それら特徴を述べる．最後に実際の RDF データを特徴に基づいて分類する．

2.2.1 実際に普及している RDF データ

以下で実際にインターネット上で配布されている RDF データの使用用途や特徴などを挙げる．

RDF Site Summary (RSS 1.0) RSS 1.0 [32] は，ウェブサイトを資源としたメタデータで，サイトに含まれる文書の見出しや要約などのリストを提供することを目的に設計された．近年ではニュースサイトや Weblog で多く用いられており，サイト内の記事が更新されるたびに RSS データも更新されるよう用いられている．そのため，利用者は RSS リーダーと呼ばれるアプリケーションを用いて，更新した記事の情報を効率的に収集することが可能になっている．利用用途が単純で，自動生成されるため非常に大量に作成されており，もっとも普及した RDF データの一つである．

RSS データのサイズは，ウェブサイトの規模や記載された記事数にも依存するが，一つの RSS 文書自体のサイズは一般的に小さい．RSS データの構造は，文書資源のメタデータを列挙しただけで単純である． さらに，他の RSS データと連携して複雑な有向グラフ構造を形成するような使用方法は現状ではほとんど見られない．

Dublin Core Dublin Core [14] は，資源のタイトルや作者，作成日などといった様々な資源に共通する 15 の基本的な情報をメタデータとして記述するための語彙を定義している． Dublin Core のみで構成される RDF データよりもむしろ，他の

RDF データと共に同時に用いられることが多い．そのため，多くの RDF フォーマットで併用されている．

Dublin Core のデータサイズは，同時に記述する側の RDF フォーマットにも依存するが基本的にデータサイズは小さく，Dublin Core データの構造は 15 の基本要素を平坦に列挙した単純な構造である． RSS と同様に他の Dublin Core データと連携して複雑な構造を形成する使用法は見られない．

Friend of a Friend (FOAF)　FOAF [11] は，友人の関係を表現するための RDF メタデータフォーマットである．すなわち，資源としての対象は，インターネット上に存在するデータを対象にしているわけではなく，実在する人物を対象としている．写真やメールアドレス，関心事，所属組織，知人などといった個人情報を表現することができる．また，個人情報を扱うために，電子署名などを用いて信頼性を高める手法を採用している．

FOAF データの特徴として，その構造は一般的に自己紹介と友人紹介のためにに使用されているために，単一の FOAF データでは単純な構造をしているが，複数の FOAF データを組み合わせると，非常に複雑な有向グラフ構造となる．データサイズは，単一の FOAF データは一般的に十分に小さい．

WordNet　WordNet [5] は，オンライン英英辞書で，英語の名詞，動詞，形容詞，および副詞を定義しており，それぞれの単語の意味を説明してある．RSS や Dublin Core， FOAF などとは異なり，利用者が用語を追加するために RDF データを記述することは基本的に想定されておらず，ただ辞書やシソーラスとして利用するだけである． WordNet の RDF 文書は，次の四つの RDF 文書から構成されている． 1) 単語を定義しただけのもの， 2) 各単語の意味を示したもの， 3) 類似した単語同士を列挙したもの，および 4) それぞれの単語に対して意味的に下位語になる単語を列挙したものである．

WordNet データの特徴に関して述べる．上記のいずれの RDF 文書も辞書データであることからわかるように，数十 MB と大きいサイズの文書である．それぞれの文書の構造は， (1) と (2) は，単純で平坦な構造をしており， (3) は同義語を示しているために，閉路を含む有向グラフ構造で， (4) は，継承関係を表してい

るために非巡回有向グラフ構造となっている．また，(1) は各単語が定義されているため，その文書のみで利用されることがあるが，一方，そのほかの (2)，(3) および (4) では各単語が URI 参照によって扱われるため，(1) と共に利用して初めて意味を成す.

Open Directory Project (ODP，dmoz とも呼ばれる)　ODP [28] とは，インターネットサイトを階層的なディレクトリ構造で表すプロジェクトで，人手で開発されているものの中では世界最大の規模である．WordNet と同様に，利用者が RDF 文書へ追加や編集することはなく，利用するのみである．また，ODP データは単一の RDF 文書であるが，データのサイズは非常に大きく，およそ 1.3 GB の容量を持つ．その構造は，まさにディレクトリ構造をなしており，ほとんど木構造となっている．木の葉に当たるサイトの URL が複数のディレクトリに含まれるために，完全な木構造にはならない．なお，最新の RDF の仕様には準拠できておらず，スキーマデータも公開されていない.

Gene Ontology　Gene Ontology [15] は，生命科学の用語を定義したオントロジーで，遺伝子産物を資源としてメタデータを付与している．生命科学の分野では，これまで生物種の違いによって独自に研究が進められていた．そのため相互のつながりはほとんど無く，同一の機能を持つタンパク質であっても異なった名で識別されていた．それらを統一し，分子の生体機能に関してゲノムデータベースを統合的利用するために，Gene Ontology が発足した．WordNet や ODP と同様に利用者によって RDF データが更新されることは無い．生物用語を表した RDF データとその情報に注釈を付与した二つの RDF 文書が提供されている.

Gene Ontology は，遺伝子産物の注釈と遺伝子産物間の継承関係とを表しており，その構造は，非巡回有向グラフ構造をしており，その深さも深い．RDF 文書のサイズは，16 MB と 200 MB で，非常に大きなサイズの RDF 文書である．なお，Gene Ontology のスキーマデータは公開されていない.

2.2.2 RDF データの分類

本節では， 2.2.1 節 で述べた RDF データの特徴について基づいて，RDF データを大まかに分類する．分類対象は，現在普及している代表的な RDF データ RSS [32]， FOAF [11], Dublin Core [14]，WordNet [5], ODP [28]，Gene Ontology [15] とする．

現在の RDF データは大まかに次の二つに分類できる． (1) 作成者は利用者で，誰でも RDF データを作成することができ，それぞれのファイルサイズは小さいという特徴を持つ RDF データと， (2) 作成者は特定の団体で，利用者が RDF データを作成することはなく，データサイズが大きいという特徴をもつ RDF データである．

RSS や FOAF, Dublin Core は (1) に属する種類の RDF データである．これらは，利用者が RDF データを作成，編集するタイプのメタデータであって，一般的にデータサイズは小さいという特徴を持っている．また， WordNet や ODP , Gene Ontology は (2) に属する種類の RDF データである．これらの RDF は利用者が RDF データを直接作成，編集することは無く，特定の団体が RDF データを作成し，これらは大きなサイズである．

RDF グラフの構造の面から RDF データを分類することを試みたが，構造の面から現在普及している RDF データを分類することは困難である．現在普及している RDF データのグラフの構造は，単一の RDF 文書では，非巡回有向グラフ構造のものがほとんどである． FOAF などのように複数の RDF 文書を組み合わせる場合では，その構造が閉路を含むことはあるが，単一文書の場合， RDF グラフの構造は多くのフォーマットの RDF で，基本的に単純な構造をしている．この理由は次の二つであると考えている．一つは現在普及している RDF フォーマットの多くが，単一の RDF データで閉路を含むことを考慮してないためである．もう一つは，現在は複雑な構造の RDF データを生成，編集するアプリケーションが存在しないためで，結局複雑な RDF データを作成するためには人によって作成しなければならない．また，人手によって作成する場合も大規模で複雑な RDF データを作成することは困難である．

第3章

関連研究

3.1 従来の RDF データベースとそれらの問題点

これまでに，いくつかの RDF データベースが提案されてきた [2], [3], [6], [36], [25], [31], [33]．本節では，従来の RDF データベースについて議論し，それぞれの手法とその問題点を挙げる．

3.1.1 従来の RDF データベース

表 3.1 に これまで提案されてきた代表的な RDF データベースの格納手法を示す．表からわかるように，従来の RDF データベースは，内部で関係データベース，あるいは Berkeley DB を利用して実装しており，基本的に flat， schema， hash アプローチのうちのいずれかの手法を採用している．以下に flat， schema， hash アプローチのデータの格納方法を具体的な例を用いながら説明する．

flat アプローチ　本論文で flat アプローチと呼んでいる手法は，関係データベースを利用した手法で， RDF データをスキーマデータとインスタンスデータを区別することなく，単一の関係表に文単位で格納する手法である．具体的に， flat アプローチのスキーマを次に示し， RDF データを格納した例を図 3.1 に示す．CHARACTER とは文字列型である．

flat アプローチのスキーマ

```
1  CREATE TABLE triple ( subject CHARACTER, predicate CHARACTER, object CHARACTER );
```

表 3.1 RDF データベースの格納手法

	格納手法	Database
RDFSuite [2]	schema, flat	RDBMS
Redland [3]	hash, flat	Berkeley DB
Sesame [6]	schema	RDBMS
Jena2 [36]	flat	RDBMS
Inkling [25]	flat	RDBMS
RDFStore [31]	hash	Berkeley DB
rdfDB [33]	hash	Berkeley DB

triple Table

subject	*predicate*	*object*
'ex:Artifact'	'rdf:type'	'rdfs:Class'
'ex:Artist'	'rdf:type'	'rdfs:Class'
'ex:Painter'	'rdf:type'	'rdfs:Class'
'ex:Painter'	'rdfs:subClassOf'	'ex:Artist'
'ex:creates'	'rdf:type'	'rdf:Property'
'ex:creates'	'rdfs:domain'	'ex:Artist'
'ex:creates'	'rdfs:range'	'ex:Artifact'
⋮	⋮	⋮
'http://www.picasso.net'	'rdf:type'	'ex:Painter'
'http://www.picasso.net'	'ex:first'	'Pablo'
'http://www.rodin.net'	'ex:first'	'Auguste'
'http://www.rodin.net'	'ex:last'	'Rodin'

図 3.1 RDF データを格納した例 (flat アプローチ)

schema アプローチ　本論文で schema アプローチと呼んでいる手法は，関係データベースを利用した手法で，RDF スキーマデータで定義されているクラスやプロパティごとに関係表を作成し，それぞれの関係表に関連した資源の URI 参照を格納していく手法である．具体的に，schema アプローチのスキーマを次に示し，RDF データを格納した例を図 3.2 に示す.

schema アプローチのスキーマ

```
1  CREATE TABLE metaclass ( URI CHARACTER, tableName CHARACTER, type CHARACTER );
2  CREATE TABLE subClassOf ( super CHARACTER, sub CHARACTER );
3  CREATE TABLE subPropertyOf ( super CHARACTER, sub CHARACTER );
4  CREATE TABLE Artist ( instance CHARACTER );
5  CREATE TABLE Painter ( instance CHARACTER );
6  CREATE TABLE Sculptor ( instance CHARACTER );
7  CREATE TABLE Artifact ( instance CHARACTER );
8  CREATE TABLE Painting ( instance CHARACTER );
9  CREATE TABLE Sculpture ( instance CHARACTER );
10 CREATE TABLE creates ( subject CHARACTER, object CHARACTER );
11 CREATE TABLE paints ( subject CHARACTER, object CHARACTER );
12 CREATE TABLE sculpts ( subject CHARACTER, object CHARACTER );
13 CREATE TABLE last ( subject CHARACTER, object CHARACTER );
14 CREATE TABLE first ( subject CHARACTER, object CHARACTER );
15 CREATE TABLE title ( subject CHARACTER, object CHARACTER );
```

hash アプローチ　本論文で hash アプローチと呼んでいる手法は， Berkeley DB を利用した手法で，RDF データをスキーマデータとインスタンスデータを区別することなく，三つの索引を作成する手法である．三つの索引は， ハッシュか B+tree で実装され，キーが主語，述語，目的語で，値が文となるように設計されている．具体的に，hash アプローチのスキーマを次に示し， RDF データを格納した例を図 3.3 に示す.

hash アプローチのスキーマ

```
1  Index subject: key = S , value = P + O;
2  Index predicate: key = P , value = S + O;
3  Index object: key = O , value = S + P;
```

S は文の主語の URI 参照を示し，P は文の述語の URI 参照を示し，O は文の目的語の URI 参照かリテラルを示す.

3.1.2　従来の RDF データベースにおける問題点

従来の RDF データベースでは幾つかの問題点を残している．それらについて本節で論述する．一つ目の問題点は，問合せ処理において経路の長さが長くなる

metaclass Table

URI	*tableName*	*type*
'ex:Artist'	'Artist'	'rdfs:Class'
'ex:Painter'	'Painter'	'rdfs:Class'
'ex:creates'	'creates'	'rdf:Property'
⋮	⋮	⋮

subClassOf Table

super	*sub*
'ex:Artist'	'ex:Painter'
'ex:Artist'	'ex:Sculptor'
'ex:Artifact'	'ex:Painting'
'ex:Artifact'	'ex:Sculpture'

subPropertyOf Table

super	*sub*
'ex:creates'	'ex:paints'
'ex:creates'	'ex:sculpts'

Artist Table

instance
'http://www.picasso.net'
'http://www.rodin.net'

Painter Table

instance
'http://www.picasso.net'

Artifact Table

instance
'http://www.picasso.net/guernica'
'http://www.picasso.net/avignon'
'http://www.rodin.net/thinker'

Painting Table

instance
'http://www.picasso.net/guernica'
'http://www.picasso.net/avignon'

creates Table

subject	*object*
'http://www.picasso.net'	'http://www.picasso.net/guernica'
'http://www.picasso.net'	'http://www.picasso.net/avignon'
'http://www.rodin.net'	'http://www.rodin.net/thinker'

paints Table

subject	*object*
'http://www.picasso.net'	'http://www.picasso.net/guernica'
'http://www.picasso.net'	'http://www.picasso.net/avignon'

title Table

subject	*object*
'http://www.picasso.net/guernica'	'Guernica Tapestry'
'http://www.picasso.net/avignon'	'Les demoiselles d' Avignon'
'http://www.rodin.net/thinker'	'The Thinker'

図 3.2 RDF データを格納した例 (schema アプローチ)

subject Index

key	*value*
'ex:Artifact'	'rdf:type' 'rdfs:Class'
'ex:Painter'	'rdfs:subClassOf' 'ex:Artist'
'ex:creates'	'rdf:type' 'rdf:Property'
'ex:creates'	'rdfs:domain' 'ex:Artist'
⋮	⋮
'http://www.picasso.net'	'rdf:type' 'ex:Painter'
'http://www.picasso.net'	'ex:first' 'Pablo'
'http://www.rodin.net'	'ex:last' 'Rodin'

predicate Index

key	*value*
'rdf:type'	'ex:Artifact' 'rdfs:Class'
'rdfs:subClassOf'	'ex:Painter' 'ex:Artist'
'rdf:type'	'ex:creates' 'rdf:Property'
'rdfs:domain'	'ex:creates' 'ex:Artist'
⋮	⋮
'rdf:type'	'http://www.picasso.net' 'ex:Painter'
'ex:first'	'http://www.picasso.net' 'Pablo'
'ex:last'	'http://www.rodin.net' 'Rodin'

object Index

key	*value*
'rdfs:Class'	'ex:Artifact' 'rdf:type'
'ex:Artist'	'ex:Painter' 'rdfs:subClassOf'
'rdf:Property'	'ex:creates' 'rdf:type'
'ex:Artist'	'ex:creates' 'rdfs:domain'
⋮	⋮
'ex:Painter'	'http://www.picasso.net' 'rdf:type'
'Pablo'	'http://www.picasso.net' 'ex:first'
'Rodin'	'http://www.rodin.net' 'ex:last'

図 3.3 RDF データを格納した例 (hash アプローチ)

にしたがって結合演算の回数が増加する点である．この問題点は，従来の RDF データベースすべてに共通して言える問題点で，もっとも重大な問題点であると理解している．また，schema アプローチにおける問題点として，スキーマを持

たない RDF データを格納することができないという点がある．さらに，flat と hash アプローチにおける問題点として，RDF スキーマデータと RDF インスタンスデータを区別無く扱うために，検索処理時において無駄な検索領域を検索する必要がある点が挙げられる．

まず，一つ目の問題点を述べる．これまでの手法では，文に基づく問合せ処理を行う場合は効率的な結果を得ることが期待できる．なぜなら，flat，schema，hash アプローチのいずれの手法も RDF グラフを文に細かく分割して，それぞれの文を格納しており，文に基づく問合せを意識した手法となっているためである．ここで，文に基づいた問合せとは，たとえば,「主語が `http://www.picasso.net` で，述語が `ex:last` であるような文の目的語を求めよ」などの一部が欠けた文を補完するような問合せである．

しかしながら，実際には，RDF データの検索の多くは，部分グラフを与え，それに一致する部分グラフを抽出するものである．すなわち，利用者による問合せでは,「資源 <`http://www.picasso.net/guernica`> のタイトルは？」などのように文に基づいた問合せよりも,「名前が `Picasso` である芸術家が描いた作品のタイトルは？」などのような複数の文で構成される部分グラフを検索する問合せが頻繁に行われる．言い換えれば，検索に用いる RDF グラフの構造は文ではなく，有向グラフ構造であることが一般的である．

このように，部分グラフによる問合せでは，比較的大きな部分グラフの発見を行う場合，従来の手法では，まず与えられた部分グラフを文に分割し，それぞれの文に問合せのキーとして検索を行い，それらの検索結果を結合して，与えられた部分グラフと一致させる必要がある．この検索結果を結合する処理を結合演算といい，基本的に処理コストが高く，性能を低下させる．この結合演算が従来の RDF データベースでは頻繁に発生する．

次に schema アプローチにおける問題点として，格納する RDF スキーマデータに基づいて関係表のスキーマを決定するために，RDF スキーマデータを持たないような RDF データを格納することができないという問題点が挙げられる．しかしながら，実際には，Gene Ontology や ODP などのスキーマを持たない RDF データも存在する．また，スキーマの変更に伴って関係表も変更するしなければ

ならないという問題点もある．

また，flat，hash アプローチにおける問題点に関して述べる．文を単一表で平坦に格納するため，スキーマ情報とインスタンス情報を区別なく格納する．このため，インスタンス情報を必要としないスキーマ情報に関する問合せにおいて，不必要な検索領域まで探索する必要が出てくる．すなわち,「`ex:Artist` クラスのすべてのサブクラスを求めよ」などといった問合せのように，スキーマ情報さえあれば処理できるような問合せでも，実際にはインスタンス情報をも含んだデータ領域から検索処理を行っている．

3.2 経路式に基づく検索手法

RDF データに対する検索として，一致する部分グラフを発見する問合せや RDF グラフを巡回するような問合せは重要である．これらの問合せは経路式に基づく問合せに変換される．すなわち，部分グラフは，複数の文で構成されるが，それらのうち，ある文の述語と別の文の主語が一致するような，連続した文は接続することで一つの経路を形成する．巡回するような問合せにおいても，同様に，連続した複数の文は経路として扱うことができる．そのため，本論文では，経路式に着目した．すなわち，問合せの構造として経路式を用いる頻度が高いことから，データを格納する場合の構造も経路式を採用することで，処理性能が向上すると考えた．

経路に基づいた検索手法は，従来から構造化文書に対する手法として多く提案されてきた． RDF データも構造化文書の一つとして扱うことができるため，従来の経路式に基づく手法を幾つか紹介する．構造化文書に対する経路式に基づく検索手法として，山本等 [50] や Qun 等 [30]， Cooper 等 [9] が提案した手法がある． 山本等の手法は，本論文の手法のアイデアの基礎となっており， XML の根から葉までの経路を文字列に置き換えて，接尾辞配列による索引を提案している．Qun 等 [30] は，これまでに McHugh 等 [22]， Milo 等 [27]，および Kaushik 等 [18] が提案した Structural Summary をより拡張した手法を提案した． Structural Summary とは，有向グラフの類似した要素を併合することで，よりコンパクトな

モデルを生成し，検索における探索領域の減少を図る手法で，併合処理の時に，経路に基づいた親子関係を比較して併合処理を行う点が特徴的である．Cooper 等 [9] は Patricia trie [19] を元にした Index Fabric という索引を提案してる．Patricia trie は大きなサイズの文字列を扱うことのできるコンパクトで効果的な索引である． Index Fabric は Patricia trie を B 木のように左右の枝をつりあわせ，半構造モデルに対応するように拡張した索引である．また，Sacks-Davis 等 [34] は，文字列が出現するまでの経路を要素名とその位置で表した素朴な手法を提案している．

経路に基づいた索引はオブジェクト指向データベースでも利用されている [4, 7, 49]． これらの手法は，継承や述語の関係を経路として扱い，問合せを高速化している． この点は本手法と類似している． 最も異なる点は，経路を表現する時，これらの手法では複数回の結合を必要とするが，本手法では文字列として経路をあつかっているため，結合をほとんど必要としない点である．

Christophides 等の研究 [8] は，RDF に対する効果的な検索を目的としている点で，本論文ともっとも関連している．彼らの手法は，これまで提案されてきた木構造のデータに対するラベリングスキームを RDF スキーマデータに適用させる方法である．具体的には，彼らの手法は，Agrawal 等 [1] が提案した各ノードの先祖数に基づいて非巡回有向グラフから全域木を生成させ，それに対してラベルをつける手法である．Christophides 等 [8] は，大まかに bit vector，prefix および interval scheme の三つに分類している． bit vector scheme [37] は，全ノード数と同数のビットを各ノードに割り当て，各ノードに対応するビットと祖先ノードに対応するビットを 1 にする手法である．prefix scheme は，Dewey Scheme [29] に代表される，親ノードのラベルを接頭辞として継承し，その後ろに，そのノードの番号をつける手法である．interval scheme [13, 1, 21] は 各ノードが (start, end) の形の interval ラベルを持ち，その値が親の interval ラベルに包含されるようなラベリングスキームである．これらの手法は，任意のノード間の接続関係の有無を確認するには適した手法ではあるが， RDF グラフを巡回するような問合せには，処理コストを要する手法である．

第4章

経路式に基づく RDF データベース

本章では，関係データベースを用いて，RDF データを経路式に基づいて格納し検索する手法を述べる．

4.1 提案手法

4.1.1 基本方針

提案する手法は，RDF データが有向辺にラベルを持つ有向グラフ構造であることに着目した手法である．まず，RDF グラフを構成するすべての文を抽出する．その後，それらを述語の種類に基づいて分類し，その分類に基づいて部分グラフへ分割する（4.1.2 節）．

本手法では，これらの分割した部分グラフに基づいて，関係表を設計しており，基本的には，それぞれ異なる部分グラフが格納される．こうすることで，部分グラフごとの特性を考慮して，関係表を設計できる．すなわち，部分グラフごとで，問合せの種類がある程度決まっており，そのうちでも発行頻度の高いものと低いものがある．発行頻度の高い問合せを効率的に処理することが，重要であるため，部分グラフごとに異なる格納方法を採用する．これによって発行頻度の高い問合せに対して効率的な検索を期待できる．

具体的には，スキーマ情報を含む部分グラフでは,「`ex:Artist` クラスのすべてのインスタンスは？」などの特定のクラスの実体を求める問合せや,「`ex:Artist` クラスのサブクラスは？」などの継承関係に基づく問合せなどが頻繁に発行され

る．そのため，RDF スキーマデータが表現する部分グラフでは，4.1.4 節 で述べるインターバルナンバリングスキームを用いる．インスタンス情報を含む部分グラフは,「ピカソ (<http://www.picasso.net/>) が作成 (ex:creates) した作品 (<http://www.picasso.net/guernica>) のタイトル (ex:title) は？」 などのような経路式に基づく問合せが頻繁に発行される．そこで，提案手法では 4.1.3 節で述べる経路式によって RDF インスタンスデータが表す部分グラフを関係表に格納する．

本論文で対象とする RDF データは，2.2.2 節 で述べた分類において， (2) のデータサイズが大きく，作成者が特定の団体であるような RDF データを対象とする．すなわち， WordNet や ODP ， Gene Ontology といった語彙や情報を体系化し，単一文書で多数の資源に対するメタデータを表現する RDF データである．また，*G* グラフ (4.1.2 節 で述べる) に閉路を含まないことも条件とする．前述したようにこのような RDF 文書は，多く Web 上に流通している．また，閉路を含むような RDF グラフであっても，強連結成分に含まれる頂点集合を縮約し，新たな一つの頂点として扱うことで，非巡回有向グラフ構造に変換することができる．この手法は今後の課題として 5.3 節 で考察する．

4.1.2 部分グラフの抽出

RDF データは， RDF Schema の仕様で定義された暗黙的な語彙の RDF メタスキーマデータと RDF Schema を用いて定義する RDF スキーマデータ，それらのインスタンスである RDF インスタンスデータで構成されている．従来の RDF データベースのうち， flat と hash アプローチでは，これらの三つを混在させた状態で扱っていた．本手法では，これらの手法とは異なり，メタスキーマデータとスキーマデータ，インスタンスデータを別々に管理する．これによって，スキーマデータのみを必要とするような問合せにおける処理で，インスタンスデータやメタスキーマデータなどの不要な情報へアクセスすることなく解を求めることができるようになる．

RDF データを分割する手法は，文の述語の種類によって RDF グラフをいくつかの部分グラフに分割する． RDF データは，すべて文によって構成されており，

すべての文は必ず何らかの述語を持っている．その述語の種類に基づいて文を分類することで，過不足なく RDF データが表す RDF グラフ全体を包括することができる．具体的には RDF データ全体から得られる部分グラフは以下の 5 種類になる．

- **クラス継承** (*CI*) **グラフ** は，クラス間の継承を表現する述語 `rdfs:subClassOf` を含む文で構成される．継承の関係を表現しているグラフで，その構造はクラスが頂点で，有向辺が `rdfs:subClassOf` である非巡回有向グラフ構造である． すなわち，*CI* グラフとは以下のような文の集合で構成される部分グラフを指す.

図 4.1 芸術家の例 (図 2.7) の CI グラフ

```
ex:Painter rdfs:subClassOf ex:Artist .
ex:Painting rdfs:subClassOf ex:Artifact ,
                            ex2:CommercialGoods .
ex:Sculptor rdfs:subClassOf ex:Artist .
ex:Sculpture rdfs:subClassOf ex:Artifact .
```

- **プロパティ継承** (*PI*) **グラフ** は，プロパティ間の継承を表現する述語 `rdfs:subPropertyOf` を含む文で構成される．このグラフもクラス継承グラフと同様に，非巡回有向グラフ構造で，プロパティが頂点となり，有向辺は `rdfs:subPropertyOf` である．すなわち，*PI* グラフとは以下のような文の集合で構成される部分グラフを指す.

図 4.2 芸術家の例 (図 2.7) の PI グラフ

```
ex:paints rdfs:subPropertyOf ex:creates .
ex:sculpts rdfs:subPropertyOf ex:creates .
```

- **タイプ** (*T*) **グラフ**は，資源とその資源のタイプを表現する述語 `rdf:type` を含む文で構成される．このグラフは閉路を含まない深さ 1 の平坦な有向グラフ構造で，頂点はクラスあるいはインスタンスであり，有向辺は `rdf:type` である．すなわち， *T* グラフとは以下のような文の集合で構成される部分グラフを指す.

図 4.3 芸術家の例 (図 2.7) の T グラフ

```
ex:Artifact rdf:type rdfs:Class .
ex:Artist rdf:type rdfs:Class .
ex:Painter rdf:type rdfs:Class ;
ex:Painting rdf:type rdfs:Class .
ex:Sculptor rdf:type rdfs:Class .
ex:Sculpture rdf:type rdfs:Class .
ex:creates rdf:type rdf:Property .
ex:paints rdf:type rdf:Property .
ex:sculpts rdf:type rdf:Property .
ex:title rdf:type rdf:Property .
ex:first rdf:type rdf:Property .
ex:last rdf:type rdf:Property .
ex2:CommercialGoods rdf:type rdfs:Class .
<http://www.picasso.net> rdf:type ex:Painter .
<http://www.picasso.net/guernica> rdf:type ex:Painting .
<http://www.picasso.net/avignon> rdf:type ex:Painting .
<http://www.rodin.net> rdf:type ex:Sculptor .
<http://www.rodin.net/thinker> rdf:type ex:Sculpture .
```

- **プロパティ定義域値域** (*DR*) **グラフ** は，プロパティの定義域を表現する述語 rdfs:domain と値域を表現する述語 rdfs:range を含む文で構成される．定義域や値域を定義する場合，主語はプロパティ，述語は rdfs:domain あるいは rdfs:range，目的語はクラスである．このため，頂点がプロパティあるいはクラス，有向辺は rdfs:domain あるいは rdfs:range の 2 種類のみである．このグラフの構造は閉路を含まない深さ 1 の平坦な有向グラフである．すなわち， *DR* グラフとは以下のような文の集合で構成される部分グラフを指す．

図 4.4 芸術家の例 (図 2.7) の DR グラフ

```
ex:creates rdfs:domain ex:Artist ;
           rdfs:range ex:Artifact .
ex:paints rdfs:domain ex:Painter ;
          rdfs:range ex:Painting .
ex:sculpts rdfs:domain ex:Sculptor;
           rdfs:range ex:Sculpture .
```

```
ex:title rdfs:domain ex:Artifact ;
         rdfs:range rdfs:Literal .
ex:first rdfs:domain ex:Artist ;
         rdfs:range rdfs:Literal .
ex:last rdfs:domain ex:Artist ;
        rdfs:range rdfs:Literal .
```

- **一般述語** (*G*) **グラフ** は，RDF データのグラフ全体から上記の 4 種類の特殊な述語を含む文を除く，残りのすべてのプロパティで構成される．このグラフは，主語と目的語を頂点とし，述語を有向辺としたグラフで，複数の種類のプロパティで構成される部分グラフとなる．このグラフは一般には閉路を含む可能性があるが，本論文では，この一般述語グラフに閉路を含まない RDF データを対象とする．実際に 2.2.1 節 で示したように，単一文書で巡回を含む RDF グラフはほとんどなく，非巡回有向グラフの RDF データが一般的である．そのため，このような制限を与えたとしても十分応用できると考えている．*G* グラフ を具体的に示すと，以下のような文の集合で構成される部分グラフを指す.

図 4.5 芸術家の例 (図 2.7) の G グラフ

```
<http://www.picasso.net>
              ex:first "Pablo" ;
              ex:last "Picasso" ;
              ex:paints <http://www.picasso.net/guernica> ,
                        <http://www.picasso.net/avignon> .
<http://www.picasso.net/guernica>
              ex:title "Guernica Tapestry" .
<http://www.picasso.net/avignon>
              ex:title "Les demoiselles d' Avignon" .
<http://www.rodin.net>
              ex:first "Auguste" ;
              ex:last "Rodin" ;
              ex:sculpts <http://www.rodin.net/thinker> .
<http://www.rodin.net/thinker> ex:title "The Thinker" .
```

RDF グラフを部分グラフに分割することで，元の RDF グラフに比べ，それぞ

れの部分グラフの構造を簡単化でき，そのグラフの特性を考慮した手法を用いて関係表に格納することができるようになる．提案する関係スキーマは，これらの部分グラフを異なる関係表で表現することを基本方針として設計されている（4.1.5 節）．

4.1.3 経路式

RDF データの構造は有向グラフ構造であるため，RDF データに対する問合せは，与えられた部分グラフと一致する部分グラフを発見する問合せや，与えられた順路をたどって到達できる要素集合を求める問合せが一般的である．これらの問合せで用いる検索のキーは経路式である．そのため，経路式に基づく問合せを効率的に処理できることが RDF データ検索の効率化に直接的につながる．そこで，提案する手法は，経路式に基づいて RDF データを格納する手法である．すなわち，問合せデータの構造と，格納される RDF データの構造を一致させることで，問合せ処理時の問合せデータの分割処理や，格納されているデータの結合処理を省くことが可能になる．

経路式に基づいて格納する対象は， RDF データ全体ではなく，特に経路式に基づいた問合せの頻度が高い *G* グラフのみを対象とする．他の部分グラフで，経路式に基づく格納を採用しない理由について述べる．*CI* グラフや *PI* グラフは，RDF スキーマデータのクラスやプロパティの継承を表現した部分グラフであるため，親子関係や先祖子孫関係を持つ要素間のつながりを知ることが頻繁に行われるが，このとき，問合せには，2 要素にはさまれた中間の要素やその数を含んでいない．すなわち，問合せの構造は経路式の形態を保っていない．そのため，任意の 2 要素間の継承関係の有無を判定できることが経路式による問合せよりも重要である．これらの処理を経路式に基づく手法で実現するには，すべての経路式に対して，与えられた 2 要素を含んでいるか否かの判定を逐次行う必要がある．次に， *DR* グラフと *T* グラフについて述べる．*DR* グラフと *T* グラフの構造は，深さ 1 の平坦な有向グラフ構造である．そのため，経路式に基づいて表現する利点がない．なぜなら，結合演算の回数を減少させるために経路式によって表現するのであることから，長い経路を含むような部分グラフでなければ，その有用性

を発揮できないからである．

RDF グラフは，頂点だけでなく有向辺にもラベルをもつ有向グラフ構造である．RDF グラフの経路を素朴に表現した場合，頂点と有向辺が交互に出現するような経路式となる．しかしながら，RDF データに対する問合せでは，頂点と有向辺が交互に出現するような経路式ではなく，一般的に，有向辺を列挙した問合せが発行される．そのため，提案する経路式に基づく格納手法は，有向辺のみを列挙した有向辺経路式を関係表に格納する手法である．以下で有向辺経路を定義する．

定義 4.1 (有向辺経路) 非巡回有向グラフ g における頂点集合を V，有向辺集合を E とする．$(v_m, v_n) \in E$ は，頂点 $v_m \in V$ から頂点 $v_n \in V$ へ向かう有向辺であるとする．

始点 v_0 から終点 v_k へ向かう有向辺経路は有向辺の有限列で表される．

$$(v_0, v_1), (v_1, v_2), \ldots, (v_{k-2}, v_{k-1}), (v_{k-1}, v_k)$$

有向辺経路の経路式は，有向辺のラベルの列で表す．

$$l(v_0, v_1), l(v_1, v_2), \ldots, l(v_{k-2}, v_{k-1}), l(v_{k-1}, v_k)$$

l は有向辺のラベルへの写像である． □

非巡回有向グラフでは，必ず入次数が 0 である頂点が一つ以上存在する．本論文では，その頂点を根と呼び，根を始点とした有向辺経路のことを絶対有向辺経路と呼ぶ．

提案手法では，G グラフに対して，すべての頂点とその頂点までの絶対有向辺経路式を抽出し，それらを関係表に格納する．すなわち，ある頂点までの絶対有向辺経路が複数存在する場合は，複数行にわたって格納する．

絶対有向辺経路式を関係表に格納することによって，G グラフ内のすべての資源への根からの経路を求めることができるようになる．しかしながら，実際に経路式に基づく問合せでは，必ずしも根からの経路式による問合せとは限らない．そのため，関係表に格納した絶対有向辺経路式を利用した問合せでは，多くの場合，問合せのキーとして与えられた有向辺経路式の先頭にワイルドカードを付加

し， LIKE 演算子を用いる SQL となる．検索条件として与えられた文字列の先頭にワイルドカードがある場合，関係データベースに組み込まれた B+tree やハッシュ関数などの索引を利用することができず，逐次検索処理となってしまう．これは，索引を用いた場合のオーダーが $O(\log n)$ あるいは，$O(1)$ であるのに対し，逐次検索では $O(n)$ となってしまうため，計算コストを要してしまう．そこで本論文では，この問題を解決するために，逆経路式による格納を採用した．すなわち，経路式上に現れる有向辺を逆順に並べ替えた逆経路式を関係表に格納する．以下に逆有向辺経路式の文法を BNF で定義する．

図 4.6 逆有向辺形路式文法 (BNF)

```
1  rearcpath ::= ( '<' predicate '@' )*
```

逆有向辺経路式の文法は，経路式内に出現する有向辺の前後に< と@ をつける．これらの記号には，有向辺，すなわち文の述語を表す URI 参照に用いられにくい文字列を用いた．また， URI 参照だけでなく， SQL で特殊な意味のある文字を利用することはできない．もし，これらの文字が有向辺を表す URI 参照に含まれていた場合は， XML における実体参照のように，特定の文字列 (&at; や <) に置き換える処理を行う．

これまで挙げてきた芸術家の例を用いると，芸術家の例の G グラフ (図 4.3) を絶対有向辺経路式を列挙すると以下のようになる．

```
1  ''
2  '<ex:first@'
3  '<ex:last@'
4  '<ex:paints@'
5  '<ex:title@<ex:paints@'
6  '<ex:sculpts@'
7  '<ex:title@<ex:sculpts@'
```

上記の有向辺経路式の 1 行目は，文字列を含んでいない．これは，根となる頂点までの絶対有向辺経路式を表している．すなわち，根から根までの経路は 0 回のステップで到達することができるために，1 回も有向辺が出現しない．図 4.5 に

示した *G* グラフの根は入次数が 0 である <http://www.picasso.net> と<http://www.rodin.net> である．そのため，それらの絶対有向辺経路式は 1 行目の '' である．また，2 行目の'<ex:first@' は "Pablo" と "Auguste" の経路式で，'<ex:last@' は "Picasso"と"Rodin" の経路式である．同様に，'<ex:paints@' は<http://www.picasso.net/guernica> と<http://www.picasso.net/avignon> の経路式で，'<ex:title@<ex:paints@' は "Guernica Tapestry" と "Les demoiselles d' Avignon" の経路式，'<ex:sculpts@' は <http://www.rodin.net/thinker> の経路式，'<ex:title@<ex:sculpts@' は "The Thinker" の経路式である．

4.1.4 ナンバリングスキームによる継承関係の表現

G グラフは，結合演算の回数を減少させるために，問合せの構造単位と格納の構造単位が一致させることを目的に，経路式に基づく格納法を採用した．一方，*CI* グラフや *PI* グラフは，経路式による問合せよりもむしろ，要素間の継承関係の有無を判定するような問合せの方が発行頻度は高い．そのため，本論文では *CI* グラフと *PI* グラフの二つの部分グラフを関係データベースに格納する際に，インターバルナンバリングスキーム [13] を採用することとした．

本節では，インターバルナンバリングスキームを非巡回有向グラフに適用する手法について述べる．インターバルナンバリングスキームとは， XML データなどの木構造のデータに対し，各頂点に数字の識別子を割り当てる手法の一つで，具体的には，深さ優先探索の前置順と後置順，および木の根からの距離の三つの数字を割り振る．この三つの数字をノード番号と呼ぶ．

ナンバリングスキームのアルゴリズムを アルゴリズム 1 に示す．このアルゴリズムは，与えられた非巡回有向グラフに対し，入次数が 0 であるすべての頂点から，深さ優先順に探索する．通常，グラフに対する深さ優先順探索では，任意の頂点を何度も通ることなく，すべての頂点を探索するが，このアルゴリズムでは，探索済みの頂点であるか否かの判定を行わないために，探索済みの頂点であったとしても，何度も訪問する可能性がある．このアルゴリズムでは，まず，ノード番号 *pre, post, depth* をグローバル変数として定義し，深さ優先順に頂点を探索

Algorithm 1: `AssignNumbers`

Data: A directed acyclic graph G

Result: G' including node numbers

```
1  V ← the set of nodes in G
2  pre ← 1
3  post ← 1
4  depth ← 0
5  foreach v ∈ V do
6  |  if the in-degree of v = 0 then
7  |  |  reAssignNumbers(v)
8  |  end
9  end
10 return G'
11 function reAssignNumbers(v) begin
12 |  pre ← pre + 1
13 |  depth ← depth + 1
14 |  Append the number pre to node v in G.
15 |  Append the number depth to node v in G.
16 |  U ← a set of nodes adjoined from v.
17 |  foreach u ∈ U do
18 |  |  reAssignNumbers(u)
19 |  end
20 |  Append the number post to node v in G.
21 |  post ← post + 1
22 |  depth ← depth − 1
23 end
```

し，深さが増すたびに *depth* を増加し，新たな頂点を探索するたびに *pre* を増加し，頂点から去るたびに *post* を増加する．したがって，各頂点では，探索される

たびにノード番号が割り振られるため，複数回探索される頂点は複数のノード番号が割符される．

図 4.7 に，このアルゴリズムを適用した例を示す．結果的に，このアルゴリズムは，非巡回有向グラフに，仮想的な根を付加し，木構造に展開して，ノード番号を割り振る手順と同様になる (図 4.7 右) .

定理 4.1 (非巡回有向グラフのノード番号数) 非巡回有向グラフ G において，唯一の根 o から頂点 v へ到達する経路数が n のとき，v に割り振られるノード番号の数は n である．

証明 非巡回有向グラフ G に対し，深さ優先順探索を行うとき，探索済みか否かの判定を行わないために，隣接頂点が存在する限り訪問する．そのため，頂点 v への訪問回数は，G の唯一の根 o からの経路数に等しい．また，ノード番号は頂点に訪問するたびに割り振られるため，頂点 v への訪問回数とノード番号数は等しい．したがって，頂点 v へ到達する経路数と v に割り振られるノード番号の数は等しい． □

インターバルナンバリングスキームを用いて割り振られたノード番号を用いた頂点間の接続関係の定理について述べる．

定理 4.2 (非巡回有向グラフに対する包含定理) 非巡回有向グラフの任意の頂点 v と u において，$(pre(v)_i, post(v)_i, depth(v)_i)$ は，v に割り振られた $i\,(1 \leq i \leq m)$ 番目のノード番号で，$(pre(u)_j, post(u)_j, depth(u)_j)$ は，u に割り振られた $j\,(1 \leq j \leq n)$ 番目のノード番号である．頂点 v が，頂点 u の先祖であるならば，次の条件を満たす (i, j) が少なくともひとつ存在する．

$$pre(v)_i < pre(u)_j \wedge post(u)_j < post(v)_i$$

このときの v と u の距離は，$|depth(u)_j - depth(v)_i|$ である．

証明 深さ優先探索の過程で，頂点に訪問するたびにその頂点をプッシュし，頂点から去るたびにポップするスタックがあるとする．すなわち，このスタックは非巡回有向グラフの唯一の頂点から訪問している頂点までの経路上のすべての頂点を順に保持している．このスタックでは，頂点 v が頂点 u の先祖であるならば，u は v より後にプッシュされ，かつ v より先にポップされる場合がある．

仮に頂点 v が頂点 u の先祖であるならば，$pre(v)_i < pre(u)_j \wedge post(u)_j < post(v)_i$ を満たす (i, j) がひとつも存在しないと仮定する．仮定より，頂点 v が 頂点 u の先祖であるならば，$pre(v)_i < pre(u)_j \wedge post(u)_j < post(v)_i$ を満たす (i, j) は一つも存在しない．$pre(v)_i < pre(u)_j \wedge post(u)_j < post(v)_i$ を満たす (i, j) が一つも存在しないのであれば，u は v より後にプッシュされ，かつ v より先にポップされることはない．矛盾が生じたため，頂点 v が頂点 u の先祖であるならば，$pre(v)_i < pre(u)_j \wedge post(u)_j < post(v)_i$ を満たす (i, j) が少なくともひとつは存在する． □

図 4.7 を用いて，定理 4.2 を適用した例を幾つか挙げる．まず， 2 頂点 A と E に関して，A のノード番号は (2,5,1) ，E のノード番号は (4,1,3) ， (6,3,3) ， (8,6,2) である．このとき，両者のノード番号のすべての組み合わせを比較する．まず，A の ノード番号と E の 1 番目のノード番号で比較すると，$2 < 4 \wedge 1 < 5$ が成り立つため，A から E に向かう長さ 2 の経路が存在する．また，E の 2 番目のノード番号で比較すると $2 < 6 \wedge 3 < 5$ が成り立つため，A から E に向かう長さ 2 の経路が他にも存在する．さらに，E の最後のノード番号で比較すると $2 < 8 \wedge 6 < 5$ は成立しないため，A から E に向かう経路はちょうど二つあることが確認できる．同様に， 2 頂点 A と D は，この手法を CI グラフと PI グラフに適用し，各頂点のノード番号を調べることで二つのクラス (プロパティ) 間に継承関係があるかどうかを容易に知ることができる．A のノード番号は (2,5,1) ，D のノード番号は (7,7,1) である．これらを定理 4.2 に基づいて比較を行うと，$2 < 7 \wedge 7 < 5$ が成り立たないために，頂点 A から頂点 D に向かう経路は存在しないことがわかる．

4.1.5 提案スキーマ

本節では，4.1.2 節 で述べたそれぞれの部分グラフに対して，経路式に基づく格納手法 (4.1.3 節) とインターバルナンバリングスキーム (4.1.4 節) を用いて，実際に RDF データを格納するため関係表について述べる．

G グラフを表現するために，経路式に基づく手法を用い，CI グラフと PI グラフを表現するために，インターバルナンバリングスキームを用いることを，これ

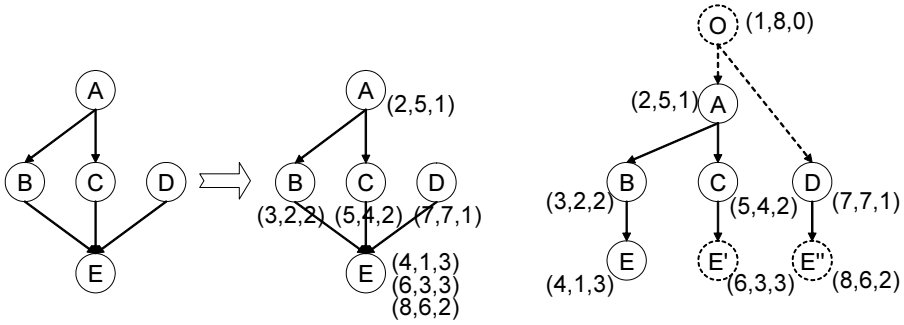

図 4.7 非巡回有向グラフに対するナンバリングスキームの適用例

まで述べてきた．残りの *DR* グラフと *T* グラフは．深さが 1 であるために，特別な手法を採用する必要が無い．

図 4.8 に提案する関係スキーマを示す．提案する関係スキーマは，六つの関係表から構成されている．なお，下線部分は，その関係表の主キーとなる属性を示している．1 行目から 4 行目は，*CI* グラフを格納するための *class* 表で，インターバルナンバリングスキームによるノード番号を表現するために，*pre, post, depth* の三つの属性を含んでいる．入次数が複数存在する頂点は，複数の異なるノード番号を持つため，複数行にわたって情報を格納する．

5 行目から 10 行目までの *property* 表は，*PI* グラフを格納することを目的とした関係表である．そのため，*class* 表と同様に，インターバルナンバリングスキームのノード番号を格納するための *pre, post, depth* の三つの属性を含んでいる．また，*property* 表は，*PI* グラフだけでなく，*DR* グラフも格納する．*PI* グラフも *DR* グラフもプロパティに関する情報を持った部分グラフであるためである．*DR* グラフは，深さが 1 であるために，特別な手法を採用したわけではなく，素朴にプロパティの定義域と値域を *domain* と *range* 属性にそれぞれ格納する．

11 行目と 12 行目の *type* 表は，*T* グラフを表現するための関係表である．*T* グラフも，*DR* グラフと同様に深さが 1 であるために，素朴に *instance* と *class* を用いて表現する．この表によってインスタンス情報とスキーマ情報の関連を表現することができる．

13 行目と 14 行目の *path* 表は *G* グラフを格納するために用いる．*G* グラフは

図 4.8 提案する関係スキーマ

```
1  CREATE TABLE class ( className CHARACTER,
2                       pre       INTEGER,
3                       post      INTEGER,
4                       depth     INTEGER );
5  CREATE TABLE property ( propertyName CHARACTER,
6                          domain       CHARACTER,
7                          range        CHARACTER,
8                          pre          INTEGER,
9                          post         INTEGER,
10                         depth        INTEGER );
11 CREATE TABLE type ( instance CHARACTER,
12                     class    CHARACTER );
13 CREATE TABLE path ( pathID  INTEGER,
14                     pathexp CHARACTER );
15 CREATE TABLE resource ( resourceName CHARACTER,
16                         ppathID      INTEGER,
17                         datatype     INTEGER );
18 CREATE TABLE triple ( subject   CHARACTER,
19                       predicate CHARACTER,
20                       object    CHARACTER );
```

経路式に基づく手法によって表現するため，*path* 表の *pathexp* 属性に，*G* グラフから得られた逆絶対有向辺経路式を列挙する．さらに，その経路式によって到達できる資源を表現するために，*resource* 表を用いる．*resource* 表の *ppathID* 属性が，*path* 表の *pathID* 属性を参照することで，資源までの経路を表現する．ある資源への絶対有向辺経路式が複数存在する場合は，そのすべての出現を異なる行として *resource* 表に格納する．さらに，*G* グラフを表現するには，経路式に基づく手法だけではすべての情報を表現しきれないため，*tirple* 表を用いて，*G* グラフを文の形式で格納する．

図 4.9 に 図 2.7 で示した芸術家に関する RDF データの例を格納した関係表を示す．ここに示した例は，簡単のためにメタスキーマデータや名前空間などの情報は省略してある．

type

instance	*class*
'r1'	'ex:Painter'
'r2'	'ex:Painting'
'r3'	'ex:Painting'
'r4'	'ex:Sculptor'
'r5'	'ex:Sculpture'

property

propertyName	*domain*	*range*	*pre*	*post*	*depth*
'ex:first'	'ex:Artist'	'rdfs:Literal'	2	1	1
'ex:last'	'ex:Painter'	'rdfs:Literal'	3	2	1
'ex:creates'	'ex:Artist'	'ex:Artifact'	4	5	1
'ex:sculpts'	'ex:Sculptor'	'ex:Sculpture'	5	3	2
'ex:paints'	'ex:Painter'	'ex:Painting'	6	4	2
'ex:title'	'ex:Artifact'	'rdfs:Literal'	7	6	1

class

className	*pre*	*post*	*depth*
'ex:Literal'	2	1	1
'ex:Artist'	3	4	1
'ex:Sculptor'	4	2	2
'ex:Painter'	5	3	2
'ex:Artifact'	6	7	1
'ex:Sculpture'	7	5	2
'ex:Painting'	8	6	2
'ex2:CommercialGoods'	9	9	1
'ex:Painting'	10	8	2

path

pathID	*pathexp*
1	''
2	'<ex:first@'
3	'<ex:last@'
4	'<ex:paints@'
5	'<ex:title@<ex:paints@'
6	'<ex:sculpts@'
7	'<ex:title@<ex:sculpts@'

triple

subject	*predicate*	*object*
'r1'	'ex:first'	'Picasso'
'r1'	'ex:last'	'Pablo'
'r1'	'ex:paints'	'r2'
'r1'	'ex:paints'	'r3'
'r2'	'ex:title'	'Guernica ..'
'r3'	'ex:title'	'Les dem..'
'r4'	'ex:first'	'August'
'r4'	'ex:last'	'Rodin'
'r4'	'ex:sculpts'	'r5'
'r5'	'ex:title'	'The Thinker'

resource

resourceName	*pathID*	*datatype*
'r1'	1	0
'r2'	4	0
'r3'	4	0
'r4'	1	0
'r5'	6	0
'Picasso'	2	1
'Pablo'	3	1
'August'	2	1
'Rodin'	3	1
'Guernica ..'	5	1
'Les demoi..'	5	1
'The Thinker'	7	1

図 4.9 芸術家の RDF データを格納した例

4.1.6 問合せ

本節では，前節で述べた提案した関係スキーマに基づいて格納した RDF データに対して検索を行うための SQL 問合せに関して述べる．これまで論述してきたように，RDF データの構造は有向グラフ構造であるため，一般的に与えられた部分グラフに一致するような検索が頻出する．そのため，経路式に基づく問合せを効率的に処理できることが重要である．また，RDF ではスキーマデータを定義することができるため，スキーマデータに基づく問合せの効率化も重要である．

次に経路式に基づいた問合せの例を示す．

例 4.1 経路式に基づく問合せ: ある人によって描かれた (ex:paints) 資源のタイトル (ex:title) を検索

```
1 SELECT r.resource
2 FROM   path AS p, resource AS r
3 WHERE  p.pathID = r.pathID
4 AND    p.pathexp LIKE '<ex:title@<ex:paints@%'
```

4.1.3 節 で述べたように，関係表に格納されている経路式は，根から資源までの経路式ではなく，資源から根までの有向辺が逆順に現れる逆絶対有向辺経路式が格納されている．このようにすることで，上記のように経路式の最後にワイルドカードを使用することができ，それによって，関係データベースの索引を利用することができる．

また，問合せにおいて，必ずしも完全な経路式が与えられない場合がある．言い換えれば，XPath [38] の “//” や正規表現の “*” などのようなものを含んだ経路式による問合せも十分に考えることができる．そのような問合せの場合，その部分にワイルドカードを用いることで，関係データベースの検索処理にすべてを任せることができる．すなわち，たとえば，次のような SQL を発行できる．

```
1 SELECT r.resource
2 FROM   path AS p, resource AS r
3 WHERE  p.pathID = r.pathID
4 AND    p.pathexp LIKE '<fisrt@%<book@%'
```

上記の SQL は，有向辺 name から 有向辺 book までを接続する有向辺の列が，任意の経路式問合せを表している．すなわち，有向辺経路式の述語を区切るために用いている “@<” の間にワイルドカードの “%” をはさむ形になる．これによって，有向辺が表す述語の URI 参照が前方一致するものがあった場合でも，正しい解のみを返すことができるようになる．すなわち，<firstname@<book@や<first@<books@ となどの経路式には一致しなくなり，<first@<author@<book@ などの有向辺 first と 有向辺 book の間を他の有向辺（この例では author）が補完したような経路式のみが一致する．

次にスキーマに基づいた問合せの例を示す．

例 4.2 スキーマ情報に基づく問合せ：資源<http://www.w3.org/2000/01/rdf-schema#Resource> の直接の親クラスを検索

```
1 SELECT c2.className
2 FROM   class AS c1, class AS c2
3 WHERE  c1.pre < c2.pre
4 AND    c1.post > c2.post
5 AND    c1.depth = c2.depth + 1
6 AND    c1.className = 'http://www.w3.org/2000/01/rdf-schema#Resource'
```

スキーマに基づく問合せでは，特定の資源の先祖子孫関係を持つ資源を発見する検索が頻繁に発行される．提案した RDF データベースでは，これらの問合せを効率的に処理するためにインターバルナンバリングスキームを採用した．しかしながら，スキーマデータの親子関係に関する情報を持つ部分グラフは *CI* グラフと *PI* グラフで，その構造は非巡回有向グラフ構造である．従来のインターバルナンバリングスキームは，本来木構造のデータに対する手法であるため，継承関係の有無の判定が通常とは異なり，本論文で定義した 4.2 によって判定する．上記の SQL は，この包含定理に基づいている．この手法では，直接の継承関係だけでなく，幾世代はなれた継承関係であっても，5 行目の *depth* の条件を変更することで簡単に対応することができる．

従来の RDF データベースでは，インターバルナンバリングスキームを用いておらず，文に基づいて格納している．そのため，直接の継承関係，すなわち直接

の親や子を検索する問合せは効率的であることが予想されるが，幾世代か離れた継承関係の有無を判定することは困難である．なぜなら，それらの距離の指定がある場合は，その数に等しい回数だけ文を結合することで処理できるが，それらの距離が不特定であった場合は，結合する回数が不明で，すべてを発見するまで再帰的に処理し続ける必要があるためである．

4.2 性能評価

本章では，提案した RDF データベースの性能評価を，スキーマ情報に基づく問合せと，経路式に基づく問合せの二種類に対して行う．

スキーマ情報に基づく問合せの性能評価を行うには，十分大きなサイズのスキーマ情報を持つ RDF データを用いて実験を行う必要がある．しかしながら，そのような RDF データは調査した範囲では存在しておらず，正当な評価を行うことはできない．そこで，本論文では代表的なスキーマ情報に基づく問合せを分類し，それらの処理が提案手法と他のアプローチにおいてどのように実現されるかを評価した．しかしながら，一般的にスキーマ情報は，インスタンスと比べ十分小さいことが多い．そのため，スキーマに関する処理は，スキーマ情報部分にのみアクセスができるか否かという点と，様々な問合せ処理に対応することができるか否かの 2 点が大きな焦点であると考えている．

また，経路式に基づく問合せは，実験によって性能評価を行う．経路長の異なる問合せが各 RDF データベースで処理される時間を評価する．

4.2.1 スキーマ情報に基づく問合せ

本節では，それぞれのアプローチにおける，スキーマ情報に基づいた問合せの処理を議論する．以下に，代表的なスキーマ情報に基づく問合せを列挙する．文法は，RDF 問合せ言語 RQL [17] による表現である．

#1 与えられたクラス（プロパティ） v の直接の子（親）を求める問合せ

```
SELECT subClassOf^(v);
```

```
SELECT superClassOf^(v);
```

#2 与えられたクラス（プロパティ） v の子孫（祖先）を求める問合せ

```
SELECT subClassOf(v);
SELECT superClassOf(v);
```

#3 与えられたプロパティ v の定義域（値域）を求める問合せ

```
SELECT domain(v);
SELECT range(v);
```

#4 与えられたクラス v を定義域（値域）とするプロパティを求める問合せ

```
SELECT @P FROM {;v}@P;
```

#5 すべての資源を求める問合せ

```
SELECT X FROM Resource{X};
```

#6 すべてのクラス（プロパティ）を求める問合せ

```
SELECT X FROM Class{X};
```

#7 すべてのリテラルを求める問合せ

```
SELECT X FROM Literal{X};
```

#8 与えられたクラス v のインスタンスの集合を求める問合せ

```
SELECT X FROM ^v{X};
```

#9 与えられたクラス v とその子孫クラスのインスタンスの集合を求める問合せ

```
SELECT X FROM v{X};
```

#10 与えられたプロパティ v を用いた文の集合を求める問合せ

```
SELECT X, Y FROM {X}^v{Y};
```

#11 与えられたプロパティ v とその子孫プロパティを用いた文の集合を求める問合せ

表 4.1 スキーマ情報に基づく問合せに対する各データベースの処理内容

	our approach	flat/hash	schema
#1.	Subsumption	Normal (Mix)	Normal (Multi)
#2.	Subsumption	Recursive (Mix)	Recursive (Multi)
#3.	Normal	Normal (Mix)	Normal (Multi)
#4.	Normal	Normal (Mix)	Normal (Multi)
#5.	Normal	N/A	Normal (Multi)
#6.	Normal	Normal (Mix)	Normal (Multi)
#7.	Normal	N/A	Normal (Multi)
#8.	Normal	Normal (Mix)	Normal (Multi)
#9.	Subsumption	Recursive (Mix)	Recursive (Multi)
#10.	Normal	Normal (Mix)	Normal (Multi)
#11.	Subsumption	Recursive (Mix)	Recursive (Multi)

```
SELECT X, Y FROM {X}v{Y};
```

#1 から #4 は，基本的なスキーマ情報に基づく問合せ，#5 から #7 は，メタスキーマ情報に基づく問合せ，#8 から #11 は資源の型に関する問合せである．RDF データベースによって，これらの問合せを処理する方針は，大きく異なる．表 4.1 に各アプローチの処理内容を示す．本節では，hash アプローチは flat アプローチと基本的な構造は同じであるため，同一のものとして考察している．表 4.1 は以下の処理の分類に基づいて記述されている．

混在データへの問合せ処理 (Mix) この処理は，対象のデータがスキーマデータとインスタンスデータが混在していることを表す．すなわち，本来スキーマデータのみあれば処理できるような問合せであったとしても，格納方針によっては，必然的にインスタンスデータにもアクセスせざるを得ない場合がある．インスタンスデータのサイズは，一般的にスキーマデータのサイズと比べると大きいため，検索処理で無駄な処理を行っていることになる．この処理は主に，hash アプローチと flat アプローチで行われる．

複数回の問合せ処理 (Multi)　この処理は，一回の処理で解を得ることができないような処理で，前の問合せの結果を基に次の問合せを生成し，その問合せを発行する必要がある．schema アプローチは RDF スキーマ情報に基づいて関係スキーマを決定するために，テーブル名は，RDF スキーマ情報に依存する．そのため，schema アプローチによる問合せ処理はすべてこの処理に分類される．以下に schema アプローチの #10 を処理する SQL を示す.

```
1  SELECT m.att0
2  FROM metaclass AS m, namespace AS n
3  WHERE n.att1 = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
4  AND m.att2 = 'Property' ;
5             Answer: 2000000000
6
7  SELECT att0 FROM t2000000000 WHERE att2 = v ;
8             Answer: 301
9
10 SELECT * FROM tp301 ;
```

このように，1) の解を 2) の問合せの関係表名として用い，さらに 2) の解を 3) で用いている．

通常の選択演算処理 (Normal)　この処理は，関係データベースの選択演算および射影演算のみで処理可能な問合せである．結合演算も必要としないため，比較的処理コストは低い．この分類に属する問合せは，flat アプローチの #1, #3, #4, #6, #8, #10 と schema アプローチの #1, #3, #4, #5, #6, #7, #8, #10 と提案手法の #3, #4, #5, #6, #7, #8, #10 である．ただし，提案手法の #4, #5, #6, #7 と flat アプローチの #4 は解に重複を含む.

再帰問合せ処理 (Recursive)　この処理は，関係データベースでの再帰問合せによって解を得る処理である．再帰する回数だけ結合演算を繰り返すことと同意であるため，コストは重くなる．この分類に属する問合せは flat アプローチの #2, #9, #11 と schema アプローチの #2, #9, #11 である．以下に flat アプローチの #2 を処理する SQL を示す.

```
SELECT f1.subject
FROM flat AS f1
WHERE f1.predicate = 'http://www.w3.org/2000/01/rdf-schema#subClassOf'
AND   f1.object IN (
    SELECT f2.subject
    FROM flat AS f2
    WHERE f2.predicate = 'http://www.w3.org/2000/01/rdf-schema#subClassOf'
    AND   f2.object IN (
          .... (
        SELECT fn.subject
        FROM flat AS fn
        WHERE fn.predicate = 'http://www.w3.org/2000/01/rdf-schema#subClassOf'
        AND   fn.object = v ))))
```

包含定理に基づく処理 (Subsumption)　インターバルナンバリングスキームによって割り振られたノード番号の包含定理を用いる処理である．この処理によって解を得る問合せは，提案手法での #1, #2, #9, #11 である．具体的な SQL は 4.1.6 節で示した．

検索不可能 (N/A)　前述したスキーマ情報に基づく問合せの内，flat アプローチでは，#5 と #7 を処理することができない．なぜなら，Resource と Literal の区別なく格納するためで，それらを区別できる仕組みを導入することでこの問題は容易に解決できる．

表 4.1 から，従来の手法では，再帰的に処理する必要がある問合せを，提案手法では包含定理に基づいた方法で処理することができることがわかる．再帰問合せの処理では，その回数だけ結合演算が必要になるが包含定理に基づいた処理では，それが不必要になる．これは，提案手法は，インターバルナンバリングスキームを用いるために，継承に基づいた問合せは他の手法と比較して有効であることを示している．もちろん，実際にデータを用いた実験による評価を行って，正しい性能を評価する必要がある．

4.2.2 経路式に基づく問合せ

本節では，実験を通して経路式に基づいた問合せを処理する性能を評価する．

実験環境

経路式に基づく問合せの実験環境について述べる．比較対象の RDF データベースは，各アプローチから一つずつ選択した． flat アプローチの Jena2 [36] ， hash アプローチは redland [3] ， schema アプローチは RDFSuite [2] を用いた．各アプローチは，格納方法が同じであるため，基本的な処理性能に大きな差はないことから，それぞれのアプローチから一つずつ開発が現在も進んでいる RDF データベースを選択した．

実験データは，単一ファイルで大規模な RDF データである Gene Ontology [15] と WordNet [24] を採用した．なお，ODP [28] は 最新の RDF の仕様に準拠していないため用いることができない．また，Gene Ontology は，スキーマ情報がないために，RDFSuite には格納することができないため，Jena2 と redland との比較実験を行う．

実験に用いた Gene Ontolory と WordNet の RDF グラフの統計値を表 4.2 に示す．これらから，Gene Ontology より WordNet の RDF グラフの方が大きいことが確認できる．注目すべき点として，G グラフの文の数に対するノードの数の割合がある．WordNet は，352,089 に対して 321,882 であるが，Gene Ontology は 240,687 に対して 147,440 である．これは Gene Ontology の方が複雑な構造であることを示している．また，次数の最大数や経路式の数の違いも Gene Ontology の方が複雑であることを示している．

次数の増減が経路数に及ぼす影響に関して考察する．入次数はグラフの頂点に向かう有向辺の数を示しており，出次数は頂点から出る有向辺の数を示している．そのため，入次数が小さく，出次数が大きいグラフは木構造に近い形をしており，経路数は葉の数に近い．逆に入次数が大きく，出次数が小さいグラフは木構造を逆にした形で経路数も根の数に近い．また，入次数が大きく，出次数の大きいグラフの経路数は，最大で $\prod_{k=0}^{d} n_k$ となる．n_k は深さが k のノード数，d は非巡回

表 4.2 実験に用いた RDF データの統計値

	Gene Ontology	WordNet
データサイズ (kByte)	18,532	32,540
文の数	258,288	451,731
文の数 (*G* グラフ)	240,687	352,089
頂点数 (*G* グラフ)	147,440	321,882
有向辺の種類 (*G* グラフ)	11	3
次数の平均 (*G* グラフ)	1.632	1.094
最大の出次数 (*G* グラフ)	1,467	29
最大の入次数 (*G* グラフ)	17,601	397
出次数の標準偏差 (*G* グラフ)	8.556	1.520
入次数の標準偏差 (*G* グラフ)	65.026	2.995
絶対有向辺経路式の種類数 (*G* グラフ)	10,000	91
最長の経路式の長さ (*G* グラフ)	17	16

有向グラフの深さを示す．そのため，入次数と出次数は累乗的に経路数を増加させる．Gene Ontology と WordNet では，次数の平均には大きな違いはないが，出次数と入次数の最大値や標準偏差が大きく違う．そのため，Gene Ontology が表すグラフの方が複雑で，多くの経路数を持つことが分かる．

実験では，問合せとして，Gene Ontology データに対しては長さが 1 から 17，WordNet データに対しては長さが 1 から 16 の経路式をそれぞれ一つずつ無作為に選択し，それらの問合せの 5 回の処理時間を平均する．一般的な問合せにおいて，経路式の長さがこれほど長い問合せは，現実的にはほぼ行われない．しかしながら，XPath の “//” などのようなあいまい検索を考慮すると，経路式の長さに制限がなくなる．そのため，実験では，それぞれの実験データの経路式の最大長まで計測することとした．

実験環境は，関係データベースとして，PostgreSQL 7.4.3 を用い，Athlon 1.4 GHz の CPU と 1024 MByte のメモリ，OS に Gentoo Linux 1.4 を搭載した計算機を用いた．また，本実験で用いた索引の詳細を付録 A に示す．

表 4.3 データベースの格納領域のサイズ（kByte）

		Gene Ontology	WordNet
提案手法	サイズ (索引含む)	4,310,328	656,776
	path 表	5,176	88
	resource 表	2,169,456	145,448
	triple 表	39,056	70,648
	class 表	0	8
	property 表	0	8
	type 表	2,240	14,440
RDFSuite	サイズ (索引含む)	–	144,616
Jena2	サイズ (索引含む)	93,840	195,400
Redland	サイズ (索引含む)	156,288	293,024

実験結果

表 4.3 に各データベースへ Gene Ontology と WordNet を格納したときの格納領域のページサイズを示す[1]．提案手法は従来の RDF データベースに比べ，格納領域のサイズが大きいことが確認できる．特に Gene Ontology では，最小の Jena 2 に比べおよそ 45 倍ものサイズを使用している．一方，WordNet では Jena 2 との比率がおよそ 3.3 倍にとどまっていることが確認できる．これは，格納するデータの特徴に伴って格納サイズが大幅に変化することを意味している．Gene Ontology の *resource* 表の格納サイズが非常に大きいことから，Gene Ontology データの複雑さのために，*resource* 表で重複が生じ，それによってデータ領域を多く必要とすることを示している．

図 4.10 に Gene Ontology，図 4.11 に WordNet をデータとして用いた場合の実験結果を示す．横軸は経路式の長さ，縦軸はその処理時間である．なお，前述したように，Gene Ontology はスキーマ情報を持っておらず，RDFSuite に格納することができないため，図 4.10 には示していない．また，グラフが途中から消えているのは，それより長い経路では，データベースがエラーを出力するために計測

[1]Redland は Berkeley DB を用いているため，ハッシュデータのファイルサイズである．

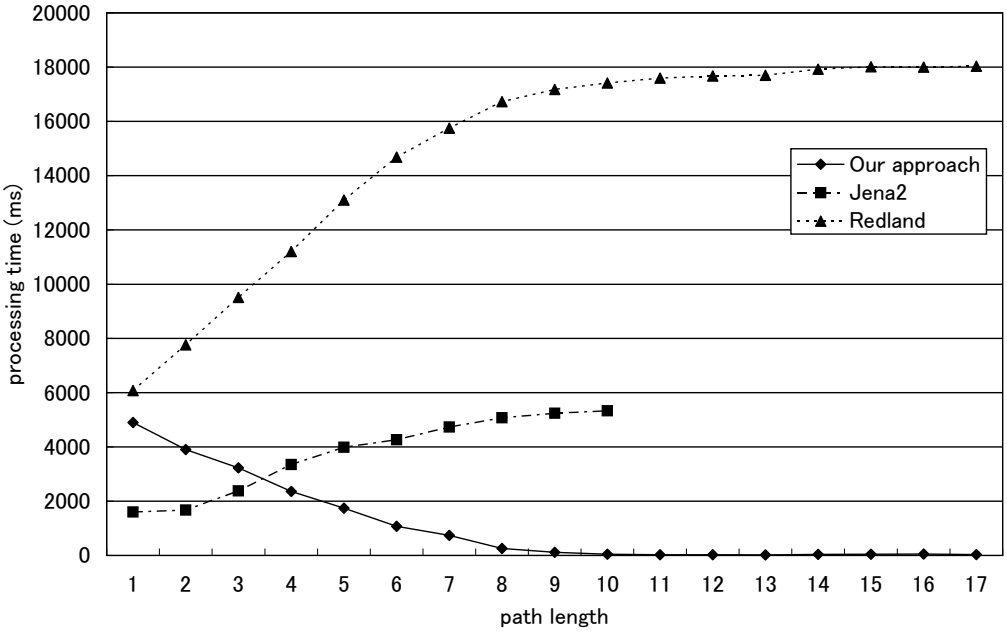

図 4.10 経路式に基づく問合せの処理時間（Gene Ontology）

することができなかったためである．

これらの図から，他の手法では，経路の長さが増すにつれて処理時間が増加しているのに対し，提案した手法では処理時間が低下していることが確認できる．これは，経路式が長くなることによって，問合せの選択度 (selectivity) が低くなり，解集合のサイズが小さくなることに起因していると思われる．また，経路の長さが 1 から 3 のような，短い場合では，提案した手法の方が性能が悪い場合があるが，本実験では *path* 表による性能のみを確認しており，このような短い経路の場合は，*triple* 表を用いることで，flat アプローチの Jena2 と同等の性能を期待できる．

表 4.3 で示したように，提案手法は従来の RDF データベースと比べ大きな格納領域を必要としている．一方図 4.10 や図 4.11 から，経路式が長い場合は非常に高速な検索を提供できる．すなわち，一般的に格納サイズと経路式の長い問合せの処理時間にはトレードオフの関係が存在しており，提案手法では格納サイズを犠牲にして処理速度を向上させた手法であると言うことができる．

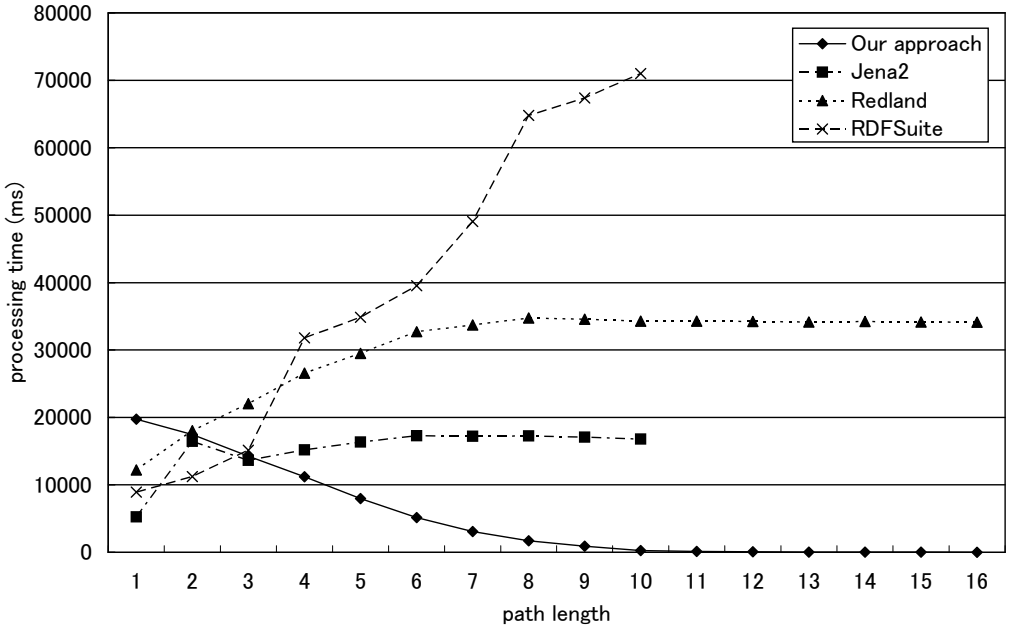

図 4.11 経路式に基づく問合せの処理時間（WordNet）

格納サイズと処理時間を同一の枠組みで比較する一般的な手法はないが，たとえば，格納サイズをハードディスクの価格，処理時間を人件費として，金額に直して比較することで，単純な目安としての比較を行うことができる．現在のハードディスクの価格は 1GByte 当たりおよそ 140 円である[2]．一方，人件費は法律で定められる最低賃金で 1 時間当たり 665 円である[3]．この金額を元にすると，Gene Ontology データの格納において提案手法と Jena2 を用いた場合，データ格納に要する金額の差はおよそ 563 円[4]で，経路式の長さが 10 の 1 回の問合せ処理時間の差を金額で表すとおよそ 1 円[5]である．格納は一回するだけでよいが，問合せ処理は問合せが発行されるたびに行う必要があるため，この場合，563 回以上の問

[2]2005 年 1 月現在販売されている 199 種類の内臓 ハードディスク IDE の 1GByte 当たりの価格を平均した．

[3]平成 16 年度地域別最低賃金額改定状況．厚生労働省のホームページ (`http://www.mhlw.go.jp/index.html`) より参照．

[4](4310328 − 93846) × (140 ÷ (1024 × 1024)) = 562.961082

[5](5329.333333 − 45.66666667) × (665 ÷ (60 × 60 × 1000)) = 0.976010648

表 4.4 経路式問合せの解の数

	Gene Ontology		WordNet	
経路式の長さ	重複なし	重複あり	重複なし	重複あり
1	15,430	248,253	123,497	908,519
2	2,112	189,692	24,871	769,281
3	984	158,845	10,069	650,465
4	480	126,763	4,728	531,501
5	248	93,540	2,354	410,398
6	134	63,221	1,196	294,886
7	73	39,030	628	187,717
8	34	21,033	335	111,179
9	20	9,185	190	60,936
10	13	2,574	107	30,562
11	8	700	61	14,981
12	5	166	40	6,727
13	3	48	22	2,666
14	1	14	16	793
15	1	8	9	208
16	1	4	4	107
17	1	4	-	-

合せを発行する場合，提案手法の方が金額的に低コストになる．563 回が多いか少ないかは利用者の主観に依存するが，データベースを利用する状況を考えると 563 回は十分少ない回数であると考えられる．

表 4.4 に問合せに用いた経路式の長さごとの解の数を “重複なし” の列に示す．提案手法では，ある頂点まで到達する経路が複数存在する場合，*resource* 表にある頂点を表すタプルを複数記述する．そのため，提案手法では，解を求める処理において，重複除去のために SQL の `DISTINCT` キーワードを用いる．この重複除去処理を行わない場合の解の数を表 4.4 の “重複あり” の列に示した．

提案手法を用いた際の処理時間が解集合のサイズに依存するか否かを確認する．表 4.4 に“重複なし”として示した解集合のサイズと処理時間との相関係数は 0.7152 であった．一方，“重複あり”として示した重複除去処理前の解集合のサイズと処理時間との相関係数は 0.9975 であった．このことから提案手法の処理時間は，解集合のサイズではなく，重複除去処理を行う前の解集合のサイズに依存していることが確認できる．すなわち，重複除去処理前の解集合のサイズが小さくなればなるほど，処理時間が短くなる．解の重複が発生する理由は，*resource* 表の *resourceName* 属性で重複が存在するためで，これは特定の資源に到達する経路が複数存在する場合に重複が発生する．すなわち，G グラフの各頂点の次数が高い場合に，資源への到達経路の数が増加することから，提案手法の経路式に基づく問合せの処理時間は，G グラフの各頂点の次数に依存していることが分かる．

4.3 条件経路式問合せのための拡張

4.3.1 条件経路式問合せ

これまで経路式に基づく RDF データベースに関して述べてきた．提案した手法によって，経路式問合せ処理における結合演算が不要になり，効率的に検索処理を行うことができるようになる．しかしながら，RDF データに対する経路式問合せには，条件によって解候補の集合をフィルタリングする問合せもある．たとえば，作成した作品のタイトルが `"Guernica Tapestry"` であるような芸術家の名前を求める問合せである．この例では，対象の頂点集合を絞り込むための条件は，複数の文の列によって表現されている．すなわち，条件として経路式 `@creates>@title>` を用い，終点が `"Guernica Tapestry"` である経路の始点集合に絞り込んでいる．このような始点，あるいは終点を特定した経路式による頂点集合をフィルタリングする問合せを条件経路式問合せと呼ぶ．

条件経路式問合せの定義を以下に示す．

定義 4.2 (条件経路式問合せ) 条件経路式問合せ $C = \langle p, U, b \rangle$ は，有向グラフ G に対する問合せとして与えられる．C は有向辺経路式 p ，条件頂点集合 U およ

び，“始点” と “終点” のいずれかの値を持つ b から成る．p は，始点集合 V_0 に含まれる頂点と終点集合 V_k に含まれる頂点を接続する有向辺経路式であるとする．b は U を始点集合として用いるか，終点集合として用いるかを示している．U は $V_0 \subset U$ あるいは $V_k \subset U$ を満たす．

- b = “始点” かつ $V_0 \subset U$ である場合，条件経路式問合せ C は順条件経路式問合せと呼ばれる．その解は終点集合 V_k である．
- b = “終点” かつ $V_k \subset U$ である場合，条件経路式問合せ C は逆条件経路式問合せと呼ばれる．その解は終点集合 V_0 である．

□

これらの条件経路式問合せは，提案した RDF データベースで処理する場合，経路式に基づいた処理では対応できない．すなわち，*path* 表に格納された逆絶対有向辺経路式を利用した検索処理を行うことができず，*triple* 表に格納された文を結合する処理を行わなければならない．

4.3.2 提案手法の拡張

本節では，これらの条件経路式問合せを処理するための手法を提案する．提案する手法は，G グラフを格納する手法を拡張した手法で，*resource* 表と *triple* 表の代わりに *headtail* 表を利用する手法である．図 4.12 に条件経路式問合せを経路式に基づく手法で処理することができる関係スキーマを示す．

4.1 節 で提案した RDF データベースと同様に *path* 表を用いる．提案した RDF データベースでは，*path* 表に逆絶対有向辺経路式を格納したが，ここで提案する手法は，*path* 表にすべての有向辺経路式を格納する．*path* 表に格納する際は，経路式を一つの文字列として格納するために，経路式を表現する文法が必要である．以下に有向辺経路式を記述するための文法を Backus Naur Form (BNF) で表す．

図 4.13 有向辺経路式文法 (BNF)

```
1  predicate ::= URIref
2  arcpath  ::= ( '@' predicate '>' )*
```

図 4.12 条件経路式問合せのための関係スキーマ

```
1  CREATE TABLE class ( className CHARACTER,
2                       pre       INTEGER,
3                       post      INTEGER,
4                       depth     INTEGER );
5  CREATE TABLE property ( propertyName CHARACTER,
6                          domain       CHARACTER,
7                          range        CHARACTER,
8                          pre          INTEGER,
9                          post         INTEGER,
10                         depth        INTEGER );
11 CREATE TABLE type ( instance CHARACTER,
12                     class    CHARACTER );
13 CREATE TABLE path ( pathID   INTEGER,
14                     pathexp  CHARACTER );
15 CREATE TABLE headtail ( head   CHARACTER,
16                         tail   CHARACTER,
17                         pathID INTEGER );
```

経路式の構文は，経路式内の有向辺の前後に@ と >を付加した．これらの記号には，有向辺，すなわち文の述語を表す URI 参照に用いられにくい文字列を用いた．また， URI 参照だけでなく， SQL で特殊な意味のある文字を利用することはできない．もし，これらの文字が有向辺を表す URI 参照に含まれていた場合は， XML における実体参照のように，特定の文字列 (&at; や >) に置き換える処理を行う．

また，4.1 節 で提案した RDF データベースでは，*resource* 表と *triple* 表を用いていたが，新たに提案する RDF データベースでは，それらに代わり *headtail* 表を用いる．*headtail* 表には，すべての 2 頂点とそれらを接続する有向辺経路式の識別子が格納される．*class* 表や *property* 表，*type* 表は 4.1 節 で提案した RDF データベースと等しい．

図 4.14 に拡張した関係スキーマに基づいて図 2.7 で示した芸術家に関する RDF データの例を格納した関係表を示す．ここに示した例は，図 4.9 と重複する *class* 表や *property* 表，*type* 表は省略している．

headtail

head	*tail*	*pathID*
'r1'	'r2'	3
'r1'	'r3'	3
'r1'	'Picasso'	1
'r1'	'Pablo'	2
'r1'	'Guernica ..'	4
'r1'	'Les dem..'	4
'r2'	'Guernica ..'	7
'r3'	'Les dem..'	7
'r4'	'r5'	5
'r4'	'August'	1
'r4'	'Rodin'	2
'r4'	'The Thinker'	6
'r5'	'The Thinker'	7

path

pathID	*pathexp*
1	'@ex:first>'
2	'@ex:last>'
3	'@ex:paints>'
4	'@ex:paints>@ex:title>'
5	'@ex:sculpts>'
6	'@ex:sculpts>@ex:title>'
7	'@ex:title>'

図 4.14 拡張手法を用いた芸術家の RDF データを格納した例

4.3.3 条件経路式の問合せ

本節では，前節で述べた拡張した RDF データベースを用いて条件経路式問合せを処理する SQL 問合せについて述べる．次に条件経路式問合せとその問合せを SQL で表現した例を示す.

例 4.3 条件経路式問合せ: タイトルが “Guernica Tapestry” である資源を描画した人物を求める条件経路式問合せ．言い換えると，終点が “Guernica Tapestry” であるような経路式 @ex:paints>@ex:title> の始点を求める問合せ.

```
1  SELECT ht.head
2  FROM   path AS p, headtail AS ht
3  WHERE  p.pathID = ht.pathID
4  AND    p.pathexp = '@ex:paints>@ex:title>'
5  AND    ht.tail = 'Guernica Tapestry'
```

この例の解は，r1 となる.

条件経路式問合せでは，複数の経路式の一方を条件として用い，一方を解を求めるために用いることができる．以下に複数の経路式を持つ問合せの例を示す.

例 4.4 条件経路式問合せ: 姓が “Pablo” である資源が描画した資源のタイトルを求める問合せ．言い換えると，終点が “Pablo” である経路式 ex:last と経路式 @ex:paints>@ex:title> の始点が一致する終点を求める問合せ.

```
SELECT ht2.tail
FROM   path AS p1, path AS p2, headtail AS ht1, headtail AS ht2
WHERE  ht1.head = ht2.head
AND    p1.pathID = ht1.pathID
AND    p1.pathexp = '@ex:last>'
AND    ht1.tail = 'Pablo'
AND    p2.pathID = ht2.pathID
AND    p2.pathexp = '@ex:paints>@ex:title>'
```

この例の解は，r1 となる.

前述の条件経路式問合せは，基礎的な問合せのみで，実際にはより複雑な問合せが発行されることが予想される．ここで述べた条件経路式問合せ処理だけでなく，4.1.6 節 で述べたスキーマに基づく問合せと組み合わせて使うことで，より実際的な問合せを発行することができる．以下に，スキーマ問合せと条件経路式を組み合わせた問合せの例を示す.

例 4.5 条件経路式問合せ: 名が “Picasso” である芸術家が描いた作品のタイトルを求める問合せ．言い換えると，終点が “Picasso” である経路式 ex:first の始点と ex:Painter クラスのインスタンス，経路式 @ex:paints>@ex:title> の始点が一致するときの終点を求める問合せ.

```
SELECT ht2.tail
FROM   path AS p1, path AS p2, headtail AS ht1, headtail AS ht2, type AS t
WHERE  p1.pathID = ht1.pathID
AND    p1.pathexp = '@ex:first>'
AND    ht1.tail = 'Picasso'
AND    ht1.head = ht2.head
AND    p2.pathID = ht2.pathID
AND    p2.pathexp = '@ex:paints>@ex:title>'
AND    p2.head = t.instance
AND    t.class = 'ex:Painter'
```

この例の解は，“Guernica Tapestry” と “Les demoiselles d’ Avignon” となる．

4.3.4 問合せ変換

本節では，RDF 問合せを SQL に変換する手法について述べる．現在，RDF 問合せ言語には，RQL [17] や RDQL [16]，SquishQL [26] などがあるが，いずれもデファクトスタンダードとして普及しておらず，各データベースで独自の問合せ言語が用いられている．本節では，具体的な問合せ言語を対象とせず，経路式問合せを表現することのできる RDF 経路式問合せグラフを RDF 問合せ言語のモデルとして提案する．その後，RDF 経路式問合せグラフを SQL へ変換する手法について言及する．

以下に RDF 経路式問合せグラフの定義を行う．

定義 4.3 (RDF 経路式問合せグラフ) RDF 経路式問合せグラフは以下の条件を満たす有向グラフ $G(V, A)$ である．

- すべての頂点は， Resource あるいは Literal のいずれかの型を持つ．
- すべての頂点は，値を持つ．値が空白であるとき，その頂点は任意の値を持つ頂点として扱われ，RDF グラフ上の任意の頂点と一致する．Resource の型を持つ頂点は，識別のための URI 参照を値として持ち， Literal の型を持つ頂点は文字列を値として持つ．Resource 型の頂点は円で描画し，Literal 型の頂点は長方形で描画する．
- 有向辺は値として識別可能な URI 参照を持つ．有向辺は矢印で描画する．値として δ を持つ頂点 n から 頂点 m を接続する有向辺を (n, δ, m) と記述する．
- Resouce の型を持つ頂点集合 V_r は，解頂点を常に一つのみ含む．解頂点である頂点に一致した頂点集合が問合せの解となる．解頂点は影付きの円で表す．

□

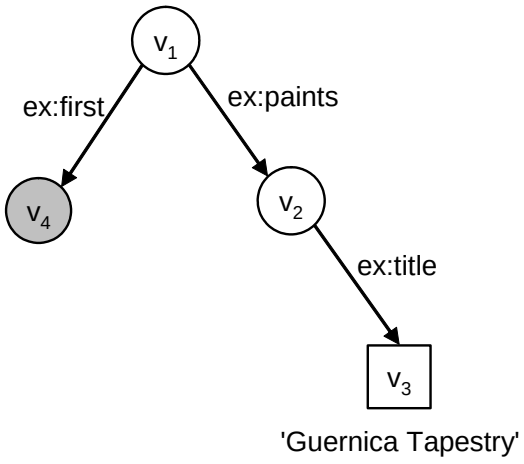

図 4.15 例 4.3 の RDF 経路式問合せグラフ

例えば，例 4.3 で挙げた例の経路式問合せを RDF 経路式問合せグラフで表すと図 4.15 のようになる．v_1 および v_2，v_4 は，型として `Resource` を持ち，値として空白を持つため，RDF グラフの任意の資源と一致する．v_3 は型として `Literal` を持ち，値として “Guernica Tapestry” を持つため，RDF グラフ上の “Guernica Tapestry” のリテラルと一致する．

RDF 経路式問合せグラフから SQL へ変換するには，まず，RDF 経路式問合せグラフから部分経路式を抽出し，部分経路式から SQL を生成する．RDF 経路式問合せグラフから経路式を抽出するアルゴリズムを アルゴリズム 2 に示す．また，部分経路式集合から SQL を生成するアルゴリズムをアルゴリズム 3 に示す.

本章では，経路式問合せを発行する対象の G グラフが非巡回有向グラフである前提があるため，アルゴリズム 2 では RDF 経路式問合せグラフの構造も非巡回有向グラフであると仮定している．アルゴリズム 2 は，与えられた RDF 経路式問合せグラフから入次数が 0 である頂点を発見し，それらの頂点から深さ優先順に頂点を探索していく，その過程で到達した頂点が値を持つか，あるいは他の経路式との分岐点と成っている場合は，それまで辿った道を経路式として格納していく．アルゴリズム 3 は，すべての部分経路式に対して，始点あるいは終点が値を持つ場合は条件として `WHERE` 句に記述し，解頂点である場合は `SELECT` 句に

Algorithm 2: `GeneratePaths`

Data: RDF 経路式問合せグラフ G
Result: 経路式の集合 P

```
1  V ← G 内の頂点の集合
2  S ← ∅ : スタック
3  P ← ∅ : 経路式の集合を格納するため
4  foreach v ∈ V do
5      if v の入次数 = 0 then
6          reGeneratePaths(v)
7  return P
8  function reGeneratePaths(v) begin
9      A ← v を始点とする有向辺の集合.
10     if v の出次数が 2 以上 then
11         path ← S から経路式 path を生成
12         P に path を追加
13         clear(S)
14     foreach a ∈ A do
15         u ← a の終点となる頂点.
16         push(a, S) /* S に有向辺 a = (v, value(a), u) を追加          */
17         if value(u) ≠ 空白 or u の入次数が 2 以上 then
18             path ← createPath(S)
19             P に path を追加
20             S ← ∅
21         if u が未探索 then reGeneratePaths(u)
```

記述する．また，他の経路式の始点，終点と一致する場合は，それらを結合する．`SELECT` 句，`FROM` 句，`WHERE` 句が生成され，それらから SQL を構築する．

Algorithm 3: GenerateSQL

Data: 経路式集合 P

Result: SQL 問合せ

```
for i ← 0 to size(P) do
    path ← P[i]
    v ← path[0] の始点頂点
    u ← path[size(path)] の終点頂点
    FROM 句に “Path p_n” を追加
    FROM 句に “HeadTail ht_i” を追加
    WHERE 句に “AND ht_i.pathID=p_m.pathID” を追加
    WHERE 句に “AND p_i.pathexp=path” を追加
    if v = 解頂点 then “ht_i.head” in SELECT 句に
    else if value(v) ≠ 空白 then
        WHERE 句に “AND ht_i.head=value(v)” を追加
    if u = 解頂点 then SELECT 句に “ht_i.tail” を追加
    else if value(u) ≠ 空白 then
        WHERE 句に “AND ht_i.tail=value(u)” を追加
    for j ← i to size(P) do
        p ← P[j]
        s ← p[0] の始点頂点
        e ← p[size(p)] の終点頂点
        if v = s then  WHERE 句に “AND ht_i.head=ht_j.head” を追加
        if v = e then  WHERE 句に “AND ht_i.head=ht_j.tail” を追加
        if u = s then  WHERE 句に “AND ht_i.tail=ht_j.head” を追加
        if u = e then  WHERE 句に “AND ht_i.tail=ht_j.tail” を追加
SELECT 句，FROM 句，WHERE 句から SQL 問合せを構築する．
return SQL 問合せ
```

4.4 本章のまとめ

本章では，大規模化しつつある RDF データのために，RDF データを格納するための経路式に基づいた関係データベースのスキーマを提案した．従来の RDF データベースにおける問題として次の二つが挙げられていた．1) 大規模 RDF デー

タに対する経路式に基づく問合せ処理において，経路長に応じた結合演算が必要になり，処理性能の低下につながる．2) schema アプローチを実装する RDF データベースでは，スキーマデータを持たない RDF データを格納することができない．また，flat と hash アプローチでは，スキーマデータとインスタンスデータを区別しないで格納しているために，スキーマに基づく問合せ処理において，インスタンスデータに対してもアクセスする必要がある．

本章では，これらの問題点を解決するための手法として，RDF データを部分グラフに分割し，それぞれの部分グラフをその特性に応じた手法を用いて，異なる関係表に格納する手法を提案した．提案手法では，部分グラフに分割して扱うことで，スキーマデータとインスタンスデータを区別して扱うことができ，それに伴って，2) の後者の問題点を解決することができる．すなわち，スキーマに基づく問合せ時に，無駄な領域を探索する必要がなくなる．本論文では，関係スキーマを固定にすることを基本方針として関係スキーマを設計した．そのため，2) の前者の問題点である，スキーマデータを持たない RDF データであっても格納することができるようになる．加えて，RDF スキーマデータに変更が起きた場合でも容易に対応することが可能である．

分割した部分グラフを異なる関係表に格納する際に，それぞれの部分グラフの特性に基づいて格納した．すなわち，インスタンスデータを含む部分グラフでは，経路式に基づいた問合せが頻出することから，結合演算の回数を減少させるために，経路式に基づく格納手法を採用した．これによって，1) の問題点を解決することができた．さらにスキーマデータでは，任意の二要素間の接続関係の有無の判定を効率化するために，インターバルナンバリングスキームを採用した．

本章では，提案した RDF データベースの性能の評価を行った．スキーマに基づく問合せに関しては，スキーマデータのサイズが十分に大きい RDF データが存在しておらず，実験による性能評価は行うことができない．RDF スキーマデータに対する問合せ処理において，無駄な検索領域を探索していないことと，十分な種類の問合せに対応していることの 2 点が重要である．主に，この 2 点について，代表的なスキーマに関する問合せを処理した場合の各データベースでの処理コストを算出した．その結果，提案したデータベースでは，無駄な領域を探索す

るような問合せはなく，処理できる問合せの種類も schema アプローチとほぼ同様であったと考えられる．

また，経路式に基づく問合せに関しては，実験によってその性能を評価した．実験では， Gene Ontology と WordNet を実験データとして用い，比較対象として Jena2 と Redland , RDFSuite を用いた．実験では，経路式が長くなるにつれて，従来の RDF データベースでは徐々に処理時間が遅くなっていくのに対し，提案した手法では逆に徐々に高速になっていった．これは，従来の手法では，経路長に応じた回数の結合演算が必要になるためである．一方，解のデータサイズは経路式が長くなると小さくなるために，提案した手法では徐々に速くなっていった．

一連の実験結果から，提案した RDF データのための関係スキーマが，大規模な RDF データに対して十分利用に耐えうることを確認した．

第5章

経路式に基づく接尾辞配列

第 4 章 では， RDF データに対する経路式に基づいた問合せを効率的に処理するための手法として，RDF データをスキーマデータとインスタンスデータとを区別して関係データベースに格納する手法を提案した．この手法によって，経路式に基づく問合せを効率的に処理することができる．またスキーマに関する問合せでは，従来の RDF データベースに比べ，スキーマデータを持たない RDF データでも格納することができ，さらに，無駄な領域を探索することなく問合せ処理を行うことができる．

RDF グラフは，頂点と有向辺を持つ有向グラフ構造をしており，頂点と有向辺の両者ともラベルを持っている．そのため，RDF グラフから生成される経路式には，有向辺のみで構成される有向辺経路式だけではなく，頂点と有向辺で構成される経路式も存在する．そのため，第 4 章 で提案した RDF データベースは，経路式に基づく問合せの一部を処理することができるが，経路式に基づいた問合せすべてを処理することはできない．すなわち，提案した RDF データベースは，有向辺経路式をもちいた問合せに関しては効率的に処理することができるが，有向辺のみではなく頂点も含んだ混合経路式による問合せは効率的に処理することができない．

本章では，上記の問題を解決するための手法として，接尾辞配列を用いた RDF データのための索引手法を提案する．

5.1 接尾辞配列を用いた RDF データ検索

5.1.1 問題提起と基本方針

RDF グラフでは，頂点だけでなく有向辺にもラベルを保持している．このため，RDF に対する経路式に基づく問合せは，有向辺のみで構成された有向辺経路式によって発行されるのみではなく，頂点も含んだ混合経路式による問合せも発行される．このような問合せに関しては，第 4 章 で提案した RDF データベースでは，効率的に問合せ処理することができない．たとえば，特定の頂点から始まる経路式によって問い合わせる場合は，有向辺経路式ではなく，混合経路式によって発行されている．

また，従来の RDF データベースでは，経路式に基づいた格納方法を採用していないため，経路式に基づく問合せの処理は，結合演算の回数が経路長に応じて増加するため，非効率的である．これらの RDF データベースにおいて，結合演算を減少させることが検索処理の効率化につながる．そのため従来の RDF データベースでは，経路式に基づいた問合せを効率的に処理することのできる索引を導入する必要がある．

本章では，これらの問題点を解決するために，経路式の接尾辞配列に基づいた索引手法を提案する．これによって，従来の RDF データベースの索引として利用することができ，さらに頂点も含む混合経路式による問合せも効率的に処理することができるようになる．

提案する手法は，提案した RDF データベースと同様に， RDF データを部分グラフへ分割する．ただし，4.1.2 節 で述べた分割とはやや異なる．具体的には，インスタンスの資源がどのクラスに属するかを表す *T* グラフは，クラスの継承関係を表す *CI* グラフに包含する．すなわち，*CI* グラフの葉にインスタンスとしての資源を付与した構造となる．このように分割したそれぞれの部分グラフからすべての経路式を抽出し，経路式のための接尾辞配列を用いることで，RDF データのための索引を構築する．

本論文で対象とする RDF データは，提案した RDF データベースと同様に，2.2.2 節 で述べた分類において， (2) のデータサイズが大きく，作成者が特定の団

体であるような RDF データを対象とする．すなわち，WordNet や ODP ， Gene Ontology といった語彙や情報を体系化し，単一文書で多数の資源に対するメタデータを表現する RDF データである．また，提案した RDF データベースと同様に，一般述語グラフ (5.1.2 節) に閉路を含まないことも条件とする．前述したようにこのような RDF 文書は，多く Web 上に流通しており，このような制約があったとしても十分応用に耐えることが出来ると考えている．また，強連結成分による縮約によって閉路を含む有向グラフから非巡回有向グラフへ変換することができる．この手法に関しては，今後の課題として 5.3 節 で考察する．

5.1.2 部分グラフの抽出

本節では，接尾辞配列を作成する前処理として，RDF グラフを部分グラフに分割する．RDF グラフから 4 種類の部分グラフを抽出する．ここで，`rdfs:subClassOf` および `rdfs:subPropertyOf`, `rdf:type` を述語とする文の関係を継承関係とよび，そのほかの述語で構成されるステートメントの関係を述語関係と呼ぶ．これらの関係に基づいて RDF グラフを分割する．RDF データからスキーマの述語関係と，インスタンスの述語関係，クラスの継承関係，プロパティの継承関係に分類し，それぞれの分類に属する有向辺で構成される部分グラフを生成する．

- **クラス継承グラフ**
  この部分グラフは，継承の関係を表現しているグラフであるため，もともと巡回を含まない非巡回有向グラフ構造である．このグラフは，クラスを頂点とし，有向辺は継承を表現する述語（すなわち，`rdfs:subClassOf` と `rdf:type` の 2 種類）で構成される．したがって，このグラフの葉はクラスのインスタンスとなることがある．すなわち，*CI* グラフと *T* グラフを混合したグラフとなる．

```
1  ex:Painter rdfs:subClassOf ex:Artist .
2  ex:Painting rdfs:subClassOf ex:Artifact ,
3                              ex2:CommercialGoods .
4  ex:Sculptor rdfs:subClassOf ex:Artist .
5  ex:Sculpture rdfs:subClassOf ex:Artifact .
```

```
  ex:Artifact rdf:type rdfs:Class .
  ex:Artist rdf:type rdfs:Class .
  ex:Painter rdf:type rdfs:Class ;
  ex:Painting rdf:type rdfs:Class .
  ex:Sculptor rdf:type rdfs:Class .
  ex:Sculpture rdf:type rdfs:Class .
  ex:creates rdf:type rdf:Property .
  ex:paints rdf:type rdf:Property .
  ex:sculpts rdf:type rdf:Property .
  ex:title rdf:type rdf:Property .
  ex:first rdf:type rdf:Property .
  ex:last rdf:type rdf:Property .
  ex2:CommercialGoods rdf:type rdfs:Class .
  <http://www.picasso.net> rdf:type ex:Painter .
  <http://www.picasso.net/guernica> rdf:type ex:Painting .
  <http://www.picasso.net/avignon> rdf:type ex:Painting .
  <http://www.rodin.net> rdf:type ex:Sculptor .
  <http://www.rodin.net/thinker> rdf:type ex:Sculpture .
```

- **プロパティ継承グラフ**
  この部分グラフもクラス継承グラフと同様に，巡回を含まない非巡回有向グラフ構造である． プロパティを頂点とし，有向辺は rdfs:subPropertyOf のみで構成される部分グラフである．プロパティはインスタンス化されないため，このグラフの葉はクラスのインスタンスを持たない．そのため，*PI* グラフ と等しいグラフとなる．

```
  ex:paints rdfs:subPropertyOf ex:creates .
  ex:sculpts rdfs:subPropertyOf ex:creates .
```

- **スキーマ述語グラフ**
  この部分グラフは，巡回を含む可能性がある．スキーマデータ内の rdfs:domain と rdfs:range によって定義された，プロパティの定義域クラスと値域クラスをグラフの頂点とし，プロパティを有向辺とするグラフである．このグラフは，*DR* グラフと同じ文の集合を基にして構成しているが，有向辺が rdfs:domain と rdfs:range ではなく，プロパティとなっているため大きく異なる．

```
ex:Artist ex:creates ex:Artifact ;
          ex:first rdfs:Literal ;
          ex:last rdfs:Literal .
ex:Painter ex:paints ex:Painting .
ex:Sculptor ex:sculpts ex:Sculpture .
ex:Artifact ex:title rdfs:Literal .
```

- **インスタンス述語グラフ**
  この部分グラフは，上記の三つの部分グラフで用いた述語以外の述語で構成される．主語と目的語を頂点とし，述語を有向辺としたグラフである．複数の意味を持つ有向辺が混在したグラフとなる．また，その構造を非巡回有向グラフ構造に制限する．このグラフは，*G* グラフと等しいグラフとなる．

```
<http://www.picasso.net>
               ex:first "Pablo" ;
               ex:last "Picasso" ;
               ex:paints <http://www.picasso.net/guernica> ,
                         <http://www.picasso.net/avignon> .
<http://www.picasso.net/guernica>
               ex:title "Guernica Tapestry" .
<http://www.picasso.net/avignon>
               ex:title "Les demoiselles d' Avignon" .
<http://www.rodin.net>
               ex:first "Auguste" ;
               ex:last "Rodin" ;
               ex:sculpts <http://www.rodin.net/thinker> .
<http://www.rodin.net/thinker> ex:title "The Thinker" .
```

上記で示したように，クラス継承グラフとスキーマ述語グラフは，提案した RDF データベースのための部分グラフとは異なるが，プロパティ継承グラフとインスタンス述語グラフはそれぞれ *PI* グラフと *G* グラフと等しい．

RDF データベースでは，*CI* グラフや *PI* グラフを表現するためにインターバルナンバリングスキームを用いた．本章で提案する手法は，すべての部分グラフで経路式に基づいた接尾辞配列を構築する手法である．これは，提案した RDF データベースと本章で提案する接尾辞配列でそれぞれが扱う処理対象の問合せが，重

複しないようにするためである．すなわち，インスタンスデータに対する問合せとして，有向辺経路式が与えられた場合は，RDF データベースで処理し，混合経路式が与えられた場合は，接尾辞配列で処理を行う．また，スキーマデータに対する問合せとして，任意の二要素間の接続関係の有無を確認する問合せが与えられた場合は，RDF データベースで処理を行い，スキーマデータの経路式に基づいた問合せが与えられた場合は，接尾辞配列によって処理を行う．

5.1.3 経路式

本節では，RDF データに対して接尾辞配列を適用するための経路式に関して述べる．本手法の目的は，頂点と有向辺が混在した経路式による問合せを効率的に処理することであるため，4.1.3 節 で述べた有向辺経路式とは異なる．

前節で述べた部分グラフから経路式を抽出するとき，インスタンス述語グラフとスキーマ述語グラフから抽出する経路式は，頂点と有向辺を交互に配置している．一方，クラス継承グラフとプロパティ継承グラフは，有向辺としての述語のラベルが `rdfs:subClassOf` と `rdf:type`，および `rdfs:subPropertyOf` のみ出現する．そのため，クラス継承グラフとプロパティ継承グラフは，有向辺を省略し，頂点のみで構成される経路式を採用した．次に，クラス継承グラフとプロパティ継承グラフの経路表現で用いるために，頂点のみで構成される頂点経路式を定義する．

定義 5.1 (頂点経路) 非巡回有向グラフ G における頂点集合を V，有向辺集合を E とする．$(v_m, v_n) \in E$ は，頂点 $v_m \in V$ から頂点 $v_n \in V$ へ向かう有向辺であるとする．

始点 v_0 から終点 v_k へ向かう頂点経路は頂点の有限列で次のように表される．

$$v_0, v_1, v_2, \ldots, v_{k-2}, v_{k-1}, v_k$$

このとき，v_{i-1} と v_i は 有向辺 (v_{k-1}, v_k) によって接続されている $(0 < i \leq k)$.

頂点経路の経路式は，頂点のラベルの列で表す．

$$l(v_0), l(v_1), l(v_2), \ldots, l(v_{k-2}), l(v_{k-1}), l(v_k)$$

l は頂点のラベルへの写像である．□

インスタンス述語グラフとスキーマ述語グラフの経路表現で用いるために，頂点と有向辺で構成される混合経路を次のように定義する．

定義 5.2 (混合経路) 非巡回有向グラフ G における頂点集合を V，有向辺集合を E とする．$(v_m, v_n) \in E$ は，頂点 $v_m \in V$ から頂点 $v_n \in V$ へ向かう有向辺であるとする．

始点 v_0 から終点 v_k へ向かう混合経路は有向辺と頂点の有限列で表される．

$$v_0, (v_0, v_1), v_1, (v_1, v_2), v_2, \ldots, v_{k-2}, (v_{k-2}, v_{k-1}), v_{k-1}, (v_{k-1}, v_k), v_k$$

混合経路の経路式は，有向辺と頂点のラベルの列で表す．

$$l(v_0), l(v_0, v_1), l(v_1), l(v_1, v_2), l(v_2), \ldots, l(v_{k-2}), l(v_{k-2}, v_{k-1}), l(v_{k-1}), l(v_{k-1}, v_k), l(v_k)$$

l は有向辺と頂点のラベルへの写像である．□

経路式の文法を定義する．以下は，BNF を用いて定義した経路を表現するための文法である．

図 5.1 経路式文法 (BNF)

```
1  instancePath ::= ('#' instance '>' '+' property '>')*
2                   ('#' instance | '"' literal '"')
3  schemaPath   ::= ('#' class '>' '+' property '>')* '#' class
4  classPath    ::= ('#' class '>')* ('$' instance | '"' literal '"')
5  propertyPath ::= ('#' property '>')* '#' property
6  class        ::= URIref
7  property     ::= URIref
8  instance     ::= URIref
```

instancePath はインスタンス述語グラフ，schemaPath はスキーマ述語グラフ，classPath はクラス継承グラフ，propertyPath はプロパティ継承グラフにそれぞれ対応している．クラス継承グラフとプロパティ継承グラフは単一の有向辺で構成されるため，その経路式は頂点経路であるのに対し，インスタンス述語グラフとスキーマ述語グラフの経路式は，混合経路である．経路式は，'>' を要

素の区切り文字として使う．また，いくつかの特殊文字は，それぞれの要素を区別するために利用する．たとえば，‘#’ は，インスタンス述語グラフの経路式では資源を示す文字で，プロパティ継承グラフの経路式ではプロパティを示す文字である．もしこれらの特殊文字が要素のラベルに含まれていれば，XML の実体参照を用いて表現する．

図 5.2 から図 5.5 は図 2.7 に示した RDF を 4 種類の部分グラフに分割し，それぞれの部分グラフから抽出した経路式の一覧である．

図 5.2 インスタンス述語経路式

```
#r1>+paints>#r2>+title>"Les demoiselles d' Avignon"
#r1>+paints>#r3>+title>"Guernica Tapesty"
#r1>+last>"Pablo"
#r1>+first>"Picasso"
#r4>+sculpts>#r5>+title>"The Thinker"
#r4>+last>"Rodin"
#r4>+first>"Auguste"
```

図 5.3 スキーマ述語経路式

```
#Artist>+creates>#Artifact>+title>#Title
#Artist>+last>#Last
#Artist>+first>#First
#Sculptor>+sculpts>#Sculpture
#Painter>+paints>#Painting
```

図 5.4 クラス継承経路式

```
#Artist>#Painter>$r1
#Artifact>#Sculpture>$r2
#Artifact>#Painting>$r3
#Artist>#Sculptor>$r4
#Artifact>#Painting>$r5
#Last>"Pablo"
#Last>"Podine"
#First>"Picasso"
#First>"Auguste"
#Title>"Les demoiselles d' Avignon"
```

```
11 #Title>"Guernica Tapesty"
12 #Title>"The Thinker"
```

図 5.5 プロパティ継承経路式

```
1 #creates>#sculpts
2 #creates>#paints
```

部分グラフが与えられたときに，経路を抽出するアルゴリズムをアルゴリズム 4 に示す．このアルゴリズムは，深さ優先順で探索を行う．手順を示す．入次数が 0 である頂点のリスト *roots*1 と入次数と出次数の差が大きい順に並び替えられた頂点のリスト *roots*2 を生成し，それらをリスト *root* に格納する．*roots* の最初の頂点から順に再帰を用いて深さ優先順探索を行う．このとき，探索の過程で訪問した頂点は，リスト *roots* から削除する．隣接頂点を持たない頂点に達したとき，探索過程で蓄積したきたスタックをもとに経路式を生成する．このアルゴリズムによって，スキーマ述語グラフを除く部分グラフからすべての経路式を抽出することができる．スキーマ述語グラフは巡回を含むため，このアルゴリズムではすべての経路式を抽出できないが，問合せ処理の際に，問合せ経路の未発見な後半の部分経路を問合せとして再発行することによってこの問題は解決できる．詳細は，5.1.5 節で例を挙げながら述べる．

5.1.4 接尾辞配列

本節では，有向グラフ構造のデータを対象とした接尾辞配列を定義し，その定義に沿って部分グラフを表現する経路式集合からそれぞれ接尾辞配列を生成する．この接尾辞配列は，通常の接尾辞配列を有向グラフに適用するように拡張したもので，直感的には，経路式中に出現する各要素を，通常の接尾辞配列における 1 文字として扱う手法である．通常の接尾辞配列は，テキストの全文検索のためのデータ構造で，具体的には，テキストからすべての接尾辞を抽出し，それらを辞書順に並べ替えた索引点の配列である．この配列に対して，二分探索を行うことで，任意の部分文字列を検索できる． 接尾辞配列の特徴は，データサイズの小ささにある．すなわち，検索に必要なデータは，テキストデータ自体と索引点の配

Algorithm 4: `CreatePaths`

```
stack : a stack to store triple
roots1 : a list of vertexes whose in-degree equal to 0
roots2 : a list of vertexes which sorted by values of (in-degree − out-degree) into
descending order − roots1
roots : a list of vertexes where roots1 + root2
foreach start ∈ roots do
    searchGraph(start)
end

function searchGraph(start : vertex) begin
    end : vertex
    arcs : set of arcs
    triple : tuple of ( vertex, arc, vertex )
    Remove the node start from list root
    arcs ← a set of arcs connected from start
    foreach arc ∈ arcs do
        end ← a vertex connected from arc
        Remove the node end from list root.
        triple ← ( start, arc, end )
        if (stack = nil or triple ∉ stack.items) then
            Push the triple triple to stack stack.
            searchGraph(end)
            Pop stack stack.
        end
    end
    path ← a string which is concatenated path and stack[0].start.
    for (i ← 0; i < stack.length; i ← i + 1) do
        path ← a string which is concatenated path and stack[i].arc, stack[i].end
    end
end
```

列のみである．

接尾辞配列を有向グラフに適用するには，複数の経路式の接尾辞を一意に指し示すための索引点に拡張を施す必要がある．そのため，2 次元の整数で構成される索引点を用いた．1 次元目は経路式の識別子，2 次元目はその接尾辞の識別子

を表す.

以下に有向グラフのための接尾辞配列および，その定義に用いる頂点経路と混合経路の接尾辞を定義する．

定義 5.3 (頂点経路の接尾辞) 有向グラフ G において，長さ m の頂点経路

$$v_0, v_1, v_2, \ldots, v_m$$

の i 番目の接尾辞は,

$$v_i, v_{i+1}, v_{i+2}, \ldots, v_m$$

である $(0 \leq i \leq m)$．すなわち，長さ m の頂点経路の接尾辞は，$m+1$ 個ある． □

定義 5.4 (混合経路の接尾辞) 有向グラフ G における，長さ m の混合経路

$$v_0, (v_0, v_1), v_1, (v_1, v_2), \ldots, (v_{m-1}, v_m), v_m$$

において，$2i$ 番目の接尾辞は,

$$(v_i, v_{i+1}), v_{i+1}, (v_{i+1}, v_{i+2}), \ldots, (v_{m-1}, v_m), v_m$$

である $(0 \leq i \leq m-1)$． $2i+1$ 番目の接尾辞は,

$$v_i, (v_i, v_{i+1}), v_{i+1}, (v_{i+1}, v_{i+2}), \ldots, (v_{m-1}, v_m), v_m$$

である $(0 \leq i \leq m)$．すなわち，長さ m の混合経路の接尾辞は，$2m+1$ 個ある． □

定義 5.5 (有向グラフのための接尾辞配列) 有向グラフ G において，識別子 j を持つ経路の i 番目の接尾辞に対する索引点を $[j, i]$ とする．有向グラフ G の接尾辞配列とは，すべての経路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である． □

図 5.6 に示した単純な有向グラフの例を用いて接尾辞配列を具体的に説明する．このグラフから得られる経路式は，混合経路である “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 5.7 のように接尾辞に二次元の索引点を付加し，図 5.8 のように辞書順に並べ替える．さらに重複した

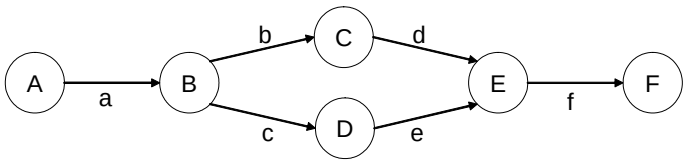

図 5.6 単純な有向グラフ

	0	1	2	3	4	5	6	7	8
0	A	a	B	b	C	d	E	f	F
1	A	a	B	c	D	e	E	f	F

図 5.7 図 5.6 の混合経路式とその接尾辞に対する索引点

経路式は，不要であるため削除する． 結果, 図 5.6 の接尾辞配列は [0,0] [1,0] [0,2] [1,2] [0,4] [1,4] [0,6] [0,8] [0,1] [1,1] [0,3] [1,3] [0,5] [1,5] [0,7] となる．また，ソートと重複要素の削除は，接尾辞配列を生成する過程で行う．つまり，索引点を格納するための空のリストを用意し，生成した接尾辞の索引点を辞書順に格納していき，その過程ですでに格納済みの接尾辞の接尾辞は格納しない．接尾辞の格納の際，その接尾辞が格納されるべき辞書順に正しい位置を検索するには，二分探索を用いて検索する．ソートおよび重複要素の削除における計算量は $O(n \log n)$ となる．

5.1.5 接尾辞配列を用いた問合せ

本節では，接尾辞配列を用いた問合せについて述べる．次に，有向グラフのための接尾辞配列を用いた問合せと解を定義する．

定義 5.6 (接尾辞配列を用いた問合せと解) 有向グラフ $G = (V, A)$ に対する接尾辞配列を用いた問合せは，経路 p によって行う．

- p の要素は，$e \in V$ あるいは $e \in A$ で構成されている．
- e_m は終点である．

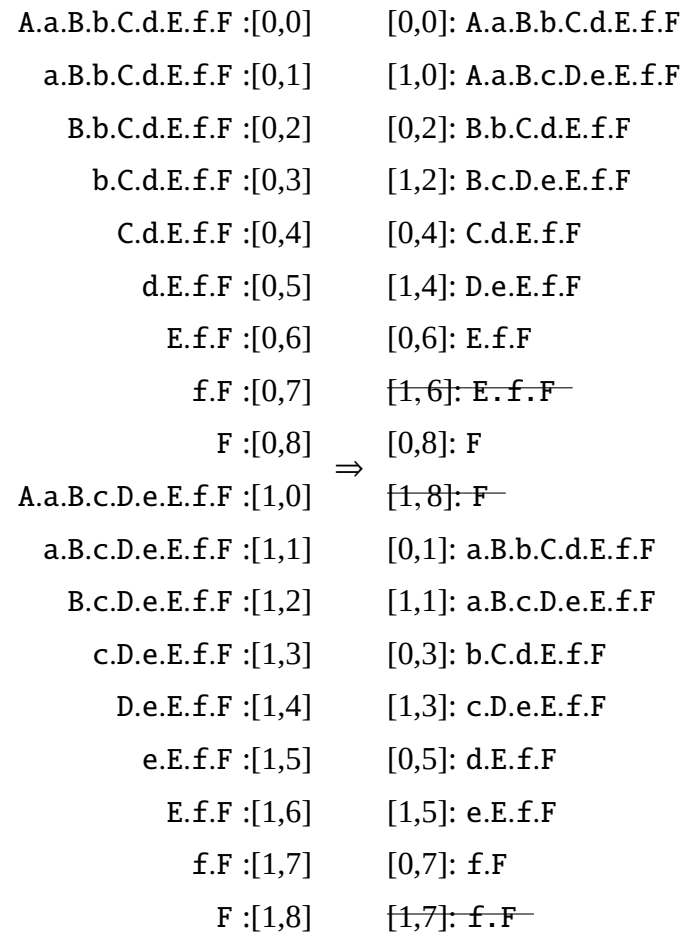

図 5.8 接尾辞のソートと重複要素の削除

- e_m と e_{m+1} を接続する有向辺 (e_m, e_{m+1}) が存在する.

を満たすとき，その解は e_{m+1} である. □

次に，問合せと RDF グラフに含まれる経路との一致に関する定義を行う.

定義 5.7 (一致および前方一致) 経路 $p = s_0, s_1, s_2, \ldots, s_n$ と $q = r_0, r_1, r_2, \ldots, r_m$ において，$s_i = r_i (0 \leq i \leq n)$ かつ $n \leq m$ のとき，p は q に一致するといい，$s_i = r_i\ (0 \leq i \leq m)$ かつ $n > m$ のとき，p は q に前方一致するという. □

問合せ処理のアルゴリズムを アルゴリズム 5 に示す．このアルゴリズムは，接尾辞配列に対して二分探索を行って目的の接尾辞を発見する手法である．解となる経路式は一つではなく，問合せ経路式に一致するすべての部分経路式を返す必要があるため，まず二分探索によって配列中の最初に一致した索引点と最後に一致した索引点（つまり配列中の一致した索引点の範囲）を求める．関数 `BinarySearch` を発行する際，引数として，配列中の最初に一致した索引点を返す場合は 0 を，配列中の最後に一致した索引点を返す場合は 1 を与える．したがって，関数 `BinarySearch` を二回発行することで，配列中の一致した接尾辞の範囲を得ることができる．検索処理の計算コストは $O(\log n)$ となる．このとき，n は配列の長さで，索引点を元に部分経路式を求めるコストは含まれていない．最後に得られた範囲から解となる部分経路式集合を求める．

このアルゴリズムはすべての部分グラフの接尾辞配列に適用できる．しかし，5.1.3 節 で述べたように，スキーマ述語グラフは，巡回を含むために，すべての経路式を抽出できない．そのためこのアルゴリズムだけでは，完全な解を得ることができない．これには次の様に対処することができる．まず，1) 問合せ経路式がこのアルゴリズム中で得られた部分経路式に対して一致ではなく，前方一致していた場合も一致したとみなす．すなわち，問合せ経路式 “a>b>c>d” は，部分経路式 “a>b” に前方一致しているとする．2) 問合せ経路式が，処理結果として得られた経路式集合に前方一致している場合は，得られた部分経路の最後の要素から始まる経路式（つまり，“b>c>d”）を新たな問合せ経路式として，再び問合せを発行する．1) と 2) を繰り返すことで巡回したグラフに対しても提案した索引を利用して検索を行うことが可能である．一つの問合せに対して繰り返し処理を行わなければならないため効率は悪いが，スキーマ述語グラフは，一般的にサイズが小さいため実用上は問題がないと考える．

5.2 性能評価

本章では，実験によって提案した接尾辞配列に基づく索引手法の性能の評価を行う．

Algorithm 5: BinarySearch

Data: suffix array *A*, path *query*, *max* is the size of suffix array, *flag* (upper bound or lower bound)

Result: the position of path *query* in suffix array *A*

```
1  begin
2      upper ← max : Int
3      lower ← 1 : Int
4      center ← 0 : Int
5      bingo ← false : Boolean
6      while lower ≤ upper do
7          center ← (Int) ((upper + lower)/2)
8          if getSuffix(A, center) = query then
9              if flag then lower ← center + 1
10             else upper ← center − 1
11             bingo ← true
12         end
13         else if getSuffix(A, center) > query then
14             upper ← center − 1
15             bingo ← false
16         end
17         else if getSuffix(A, center) < query then
18             lower ← center + 1
19             bingo ← false
20         end
21     end
22     if not bingo then
23         if flag then center ← center − 1
24         else center ← center + 1
25     end
26     return center
27 end
```

5.2.1 実験環境

実験では，比較対象として flat アプローチの RDF データベースである Jena2 [36] を採用した．本来であれば，RDF データのための索引手法と比較すべきであるが，われわれが知る限り RDF データのための索引手法は，Christophides 等 [8] が提案したラベリングスキームしかない．しかしながら，彼らの手法は経路式に基づく問合せを効率化することを目的とした手法ではなく，任意の 2 要素間の接続関係の有無の判定を効率化するための手法である．そのため，直接比較することは難しい．

実験データとして，遺伝子産物のためのメタデータである Gene Ontology [15] を採用した．Gene Ontology は分子の生体機能に関してゲノムデータベースを統合的利用するために作成されたオンラインデータベースで，主に関連する分子間の関係を表現している RDF 文書と分子の生体機能に関して記述している RDF 文書の 2 種類あり，前者は後者のサブセットである．本論文では前者を実験に採用し，データサイズを 500 KB にしたもので評価を行った．また，Gene Ontology には RDF スキーマは定義されていないため，XML の DTD を元に独自に作成した．

表 5.1 に実験で用いた問合せとしての経路式を示す．つまり，この経路式に一致する部分経路式を Gene Ontology の RDF データから生成した接尾辞配列を用いて発見することを問合せ処理とする．なお，表 5.1 では，見やすくするためそれぞれの資源の名前空間を省略してあるが，本来の経路式では含まれる．これらの経路式を問合せとして用いることで，次のような意図を含んでいる．問合せ 1 と 2 はクラスの継承関係に関する問合せ，問合せ 3 はプロパティの継承関係に関する問合せ，問合せ 4 から 6 はスキーマの述語関係に関する問合せ，問合せ 7 と 8 はインスタンスの述語関係に関する問合せである．

実験方針としては，Jena2 と提案手法の処理速度を比較を行う．まず，Jena2 の API を利用して構文解析を行い，生成された RDF データから 4 種類の部分グラフの接尾辞配列を生成した．その後，表 5.1 に示した問合せ経路式を，生成した接尾辞配列から発見するまでに要する処理時間と，それと同義の問合せを RDQL [16] を用いて Jena2 に対して発行したときの処理時間を比較した．それぞれ 100 回問合せを発行し，それらの平均をとった．

表 5.1 性能評価に用いた問合せとしての経路式

	クラス継承グラフへの問合せ	
#1	#Resource>#	サブクラスの検索
#2	#Resource>#Class>#Datatype>$	子孫クラスの検索
	プロパティ継承グラフへの問合せ	
#3	+seeAlso>#	定義域の検索
	スキーマ述語グラフへの問合せ	
#4	#Resource>+	述語の検索
#5	+dbxref>#	目的語の検索
#6	#Resource>+isDefinedBy>#Resource>+dbxref> #Resource>+member>#Resource>+seeAlso>#Resource>+	長い経路式
	インスタンス述語グラフへの問合せ	
#7	#GO:0003674>+n_associations>"	ステートメント検索
#8	#GO:0050646>+is_a>#GO:0050543>+is_a>#GO:0050542> +is_a>#GO:0005504>+is_a>#GO:0008289>+dbxref> #∅>+database_symbol>"	長い経路式

また，実験には，Athlon 1.4 GHz の CPU と 1024 MB のメモリを搭載した計算機を用い，OS は RedHat Linux 9.0 で行った．

5.2.2 実験結果

表 5.2 に Gene Ontology の RDF 文書から生成した経路式数とその接尾辞数を示す．この表からわかるように，インスタンス述語グラフでは，接尾辞数が経路数よりも少ない．これは，重複した接尾辞を削除するためで，多重辺を多く含むグラフであることが確認できる．また，クラス継承グラフやプロパティ継承グラフは非常に小さなグラフであるにもかかわらず，スキーマ述語グラフは，巡回を含むためにその値は大きくなっている．

表 5.2 経路式数と接尾辞数

	経路式数	接尾辞数
クラス継承グラフ	203	608
プロパティ継承グラフ	25	26
スキーマ述語グラフ	6,530	16,322
インスタンス述語グラフ	87,801	67,706

図 5.9 に Jena2 と提案手法の処理時間を示す．また，表 5.1 に示した問合せのうち，問合せ 2，6，8 は RDQL では表現できないため，空欄になっている．図 5.9 から判断できることは，提案した手法の方が高速であるのは問合せ 1 と 3 で，他の問合せでは， Jena2 の方が高速であった．しかしながら，問合せ 2，6，8 を RDQL で表現できれば，提案手法の方が効果的であることが予測できる．その理由を次に示す． Jena2 は，どの検索もほぼ一定の処理時間であり，これは， RDQL で表現できた問合せは，経路の長さが 1 以下のものであるため，一定の処理時間になったと考えられる．言い換えれば，仮に RDQL で長い経路式の問合せを表現できれば， Jena2 の処理速度は，現在の処理速度よりも遅くなることは明確である．一方，提案手法でも同じ部分グラフに対して問合せを発行する場合は，一定の検索速度であることがわかる．これは，同じ部分グラフから生成された同じ接尾辞配列を用いて問合せを行っているためである．このことは，提案手法での処理速度は，対象の部分グラフから抽出された経路式の数とその配列の長さにのみ依存し，問合せ経路式の長さに依存しないことを示している．したがって，問合せ 2 と 6 ， 8 において， RDQL で表現できれば，提案手法の方が効果的になることが予測できる．

5.3 閉路を含む RDF グラフへの対応方針

本章では，第 4 章で提案したデータベースでは効率的な検索を提供できない頂点を含む経路式問合せを効率的に検索するために，接尾辞配列に基づいた索引を提案した．しかしながら，経路を抽出する対象の部分グラフは閉路を含まない非

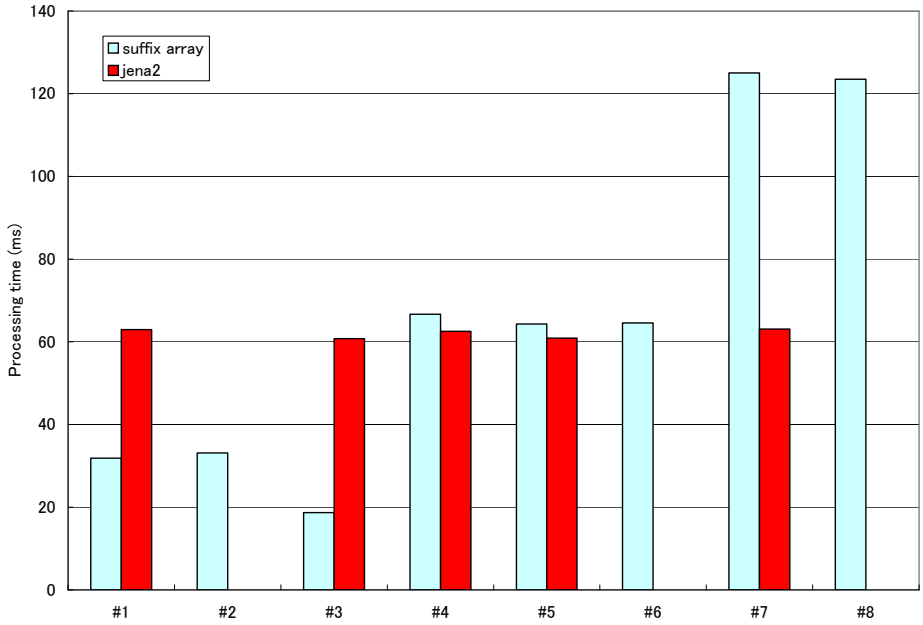

図 5.9 RDF データのための接尾辞配列と Jena2 の処理時間

巡回有向グラフとする制限を与えている．また，第 4 章で提案したデータベースでも，同様に制限がある．これは，閉路を含むことで経路数が無限になることを防ぐためである．本節では，将来の課題としての閉路を含む RDF グラフへの対応に関する一つのアイデアとして，強連結成分を用いた非巡回有向グラフ化に関して述べる．

現在，インターネット上に存在する RDF データの多くは，閉路を含んでいないため，このような制限を与えたとしても，運用上，大きな問題にはならないと考えている．しかしながら，RDF では閉路を含む有向グラフを表現できることは事実である．そのため，将来の課題として，閉路を含む RDF グラフへの対応が挙げられる．

5.3.1 強連結成分を用いた非巡回有向グラフ化

RDF データは，複数の文を用いることで，その構造は有向グラフ構造となる．RDF グラフでは，閉路を含むことができるため，より複雑な情報を表現することができる．本論文では，これまで 第 4 章 および 5.1 節 で RDF データから抽出した部分グラフの一部には閉路を含まないよう制限を与え，議論を進めてきた．本節では，将来の課題として，閉路を含む RDF グラフへの対応の一つのアイデアとして，強連結成分を用いた手法を述べる．

本方針では，まず，与えられた RDF グラフからこれまで述べてきたように，幾つかの部分グラフを抽出する．その後，経路式を生成する対象の部分グラフに対して，最大強連結成分を求め，成分に含まれる頂点集合を単一の頂点に縮約する．これによって閉路を含むグラフから非巡回有向グラフが生成される．この手法は，[1] で提案された手法をアイデアの基礎としている．

次に最大強連結成分とグラフの縮約の定義を行う．

定義 5.8 (最大強連結成分) 有向グラフ $G = (V, E)$ の最大強連結成分は，V 中の全ての頂点対 u と v において，u から v への経路と v から u への経路の両方を持つような最大頂点集合 U である．言い換えれば u と v は 互いに到達可能である．□

定義 5.9 (グラフの縮約) 有向グラフ G の辺 e を一つの頂点に縮める操作を縮約とよび，その結果得られるグラフを辺 e による G の縮約グラフ，あるいは商グラフとよぶ. □

本手法は，以下の定理を利用して，有向グラフ構造を非巡回有向グラフに変換して扱うことを基本方針としている．

定理 5.1 (最大強連結成分の縮約) 有向グラフ $G = (V, E)$ の最大強連結成分 U を一つの頂点に縮約したグラフ G_c の構造は，非巡回有向グラフ構造となる．

証明 最大強連結成分 U に含まれる頂点同士は，互いに到達可能で，$U^c = V \setminus U$ に含まれる頂点との間は強連結していないため互いに到達可能ではない．そのため最大強連結成分によって縮約された縮約グラフ G_c における任意の頂点 v_c と u_c

は強連結していない．すなわち，互いに到達可能ではないため閉路を持たない．よって，G_c の構造は非巡回有向グラフ構造である． □

上記の定理に基づいて非巡回有向グラフを生成するが，そのためには，最大強連結成分を特定する必要がある．強連結成分を求めるアルゴリズムはこれまで幾つか提案されている．具体的には，[10] では最大強連結成分の発見を計算量 $O(n)$ のコストで行う深さ優先探索に基づいたアルゴリズムや，縮約処理を計算量 $O(n)$ で行うアルゴリズムを紹介している．

この手法によって閉路を含む有向グラフを非巡回有向グラフに変換できる．この手法を RDF データに適用することで，第 4 章 や 5.1 節 で提案した手法を利用できると考えている．しかしながら，この手法を採用することで，強連結成分に含まれる頂点に対する問合せが困難になる．本章では今後の課題の方針として，強連結成分による縮約に関して述べたが，今後の課題として，強連結成分に含まれる頂点に対する問合せ処理に関して研究していく必要がある．また実装を行い本手法の性能評価を行う必要がある．

5.4 本章のまとめ

本章では，経路式の接尾辞配列に基づいた RDF データのため索引手法を提案した．第 4 章 で RDF データのための経路式に基づく関係スキーマを提案したが，提案した経路式に基づく格納方法は，有向辺のみで構成される有向辺経路式を格納することで，特定の資源までの述語の列を結合演算をすることなく発見することができる手法であった．しかしながら， RDF データに対する経路式に基づく問合せのすべてをこの手法で解決することができるわけではなく，たとえば，特定の資源から始まる経路式を発見するような問合せには，この手法を用いることができなかった．

加えて，従来の RDF データベースに格納されている RDF データに対して経路式に基づく問合せを発行すると，結合演算が増加してしまうために，そういった問合せを効率化することができる索引手法が必要であった．

本章で提案した経路式の接尾辞配列に基づく索引手法を用いることで，これら

の問題点を解決することできた．提案した手法は，まず与えられた RDF データからプロパティの定義域と値域の情報を含むスキーマ述語グラフとインスタンスにおける文で構成されるインスタンス述語グラフ，クラス間の継承関係とインスタンスとクラスの実体化の関係を表したクラス継承グラフ，プロパティの継承関係を表したプロパティ継承グラフの四つの部分グラフを抽出した．その後，それらから頂点と有向辺からなる混合グラフを生成し，経路式の接尾辞配列を構築した．この接尾辞配列が RDF データのための索引となる．この索引は，辞書順に並べ替えられているため，検索の際には二分探索による文字列マッチングを行うことで検索することができる．これによって，結合演算の回数を減少させ，さらに，計算量の小さい手法で検索することができる．

本章では，提案した索引手法の性能を評価するために実験を行った．実験では，Jena2 を比較対象とし，実験データとして Gene Ontology を用いた．本来であれば， 他の RDF データのための索引手法との比較を行うべきであるが， RDF データのための索引手法は現時点ではあまり提案されておらず，すでに提案されている手法は経路式問合せを効率化することを目的として提案されているわけではない．実験結果は， Jena2 で採用されている RDF 問合せ言語の RDQL が混合経路式を表現できないために， Jena2 の混合経路式の性能を正確には評価できなかった．しかしながら，もし RDQL で混合経路式を表現することができれば，提案手法が勝るはずである．なぜなら， Jena2 は flat アプローチを採用しているために経路式が長くなるにつれて必ず，処理コストがかかってくるが，提案した手法での処理時間は，経路長には依存せず，対象のデータサイズにのみ依存していることが確認できたためである．

実験を通して，提案した接尾辞配列に基づく索引手法によって，混合経路式による問合せを効率的に処理することを確認した．また，本手法を他の RDF データベースの索引として利用することで，経路式に基づく問合せ時に発生していた結合演算の回数が増加することで起こる処理性能の低下という問題を解決することができるようになる．

第6章

おわりに

6.1 本論文のまとめ

本論文では，大規模な RDF データのための効率的な検索のための手法を提案した．RDF データを関係データベースに格納する手法と経路式の接尾辞配列を用いた索引手法である．

従来の RDF データのための検索では，いくつか問題点があった．具体的には，1) RDF データの検索では，与えられた部分グラフを発見するような問合せが頻出するため，そのような問合せの場合，結合演算の回数が増加してしまい，大規模な RDF データでは処理時間が大幅にかかってしまうという点．2) schema アプローチでは，RDF スキーマデータに基づいて関係スキーマを決定するため，RDF スキーマデータを持たない RDF データを扱うことができないという点，3) hash, flat アプローチではスキーマデータとインスタンスデータを区別なく単一の関係表に格納するために，スキーマに関する問合せを処理する際に，無駄な領域までも探索しなければならない点である．これらを解決することを目的として，RDF データを関係データベースに格納する手法を提案した．

提案した関係スキーマでは，インスタンスデータとスキーマデータを区別して格納することと，RDF のスキーマに依存しない関係スキーマにすることを基本方針とし設計した．そのための手法として，RDF データを部分グラフに分割し，それぞれの部分グラフをそれぞれの特性に基づいて適切な手法を用いて，異なった関係表に格納する手法を採用した．具体的に，スキーマデータを含む部分グラフでは，二要素間の接続関係の有無を判定するために，インターバルナンバリン

グスキームを用いて格納することで，不要な頂点へアクセスする必要がない．また，インスタンスデータを含む部分グラフでは，結合演算の回数を減少させるために，有向辺経路式に基づいて格納した．

スキーマに関する問合せ処理では，提案手法は，従来の RDF データベースと比べ，無駄な探索領域へアクセスする必要がないことを確認した．また，経路式に基づく問合せの性能評価は実験によって行った．従来の手法では，経路式が長くなるにつれ，処理時間が遅くなっていくのに対し，提案した手法では，処理時間が短くなっていった．これは，経路式が長くなることによって，問合せの選択度が低下し，解集合のサイズが小さくなることが起因している．

提案した RDF データベースのための関係スキーマでは，有向辺経路式を採用して格納しているが，RDF データに対する経路式に基づく問合せは必ずしも，有向辺経路式のみが発行されるわけではなく，頂点を含む経路式が発行されることもある．そのため，頂点を含む経路式の問合せ処理を効率化することを目的とした接尾辞配列に基づく索引手法を提案した．また，この手法は従来の RDF データベースの索引として利用することができるため，従来の RDF データベースに対する経路式問合せの処理効率の向上につながる．

提案した索引手法は，RDF データを部分グラフに分割し，それらから頂点と有向辺で構成される混合経路式と頂点のみで構成される頂点経路式を抽出し，経路式に含まれる頂点名や有向辺名を 1 文字とし，経路式を文字列に見立て，接尾辞配列を構築した．この接尾辞配列は，経路式の識別子と接尾辞の番号からなる索引点の列が，経路式の接尾辞の辞書順に並べ替えられているため，与えられた経路式と二分探索による文字列比較を行うことで，目的の経路式の接尾辞を容易に発見することができる．本論文では，提案した索引手法の性能を実験によって評価した．実験で比較対象として用いた Jena2 の問合せ言語では，混合経路式を表現できないという問題があるために，すべての問合せでの比較ができなかったが，経路式を表現することができれば，提案した接尾辞配列による手法が勝ることを実験結果から予測できた．それは，Jena2 の処理時間は経路長と RDF データ全体のサイズに依存していたが，提案手法では，経路長には依存せず， 分割された部分グラフのサイズのみに依存していたためである．

これらから，本論文で提案した二つの手法は，大規模な RDF データを扱う上で有用な手法であること確認した．

6.2 今後の課題

今後の課題として，閉路を含む RDF データへの対応や RDF データのための問合せ処理，RDF と RDF Schema の仕様への準拠などがあげられる．

閉路を含む RDF データへの対応に関しては，本論文で提案した手法では，一部の部分グラフで閉路を含まないよう制約を与えた．この問題を解決する方針として，5.3 節 で述べた強連結成分を用いた手法がある．この手法によって問題が完全に解決するわけでなく，新たに強連結成分に含まれる頂点に対する問合せをどのように処理するかという問題が発生する．

問合せ処理に関しては，与えられた RDF 問合せ言語から SQL への変換や，問合せに応じた索引の利用，与えられた問合せの最適化などがあげられる．RDF データのための問合せ言語もすでにいくつか提案されているが，未だ標準的な言語は出現していない．そのため，RDF データに対する問合せを網羅的に用意し，それらに対する処理を考察する必要がある．注意すべき点として，RDF は Semantic Web で用いられることを考慮し，レイヤーケーキのように問合せ処理に関しても段階的に扱うべきであると考えている．

本論文で提案した手法が対象としている RDF や RDF Schema のモデルは簡単のために単純化している．今後は仕様へ準拠することが必要となる．具体的には，RDF では，集約を扱うための Bag や Alt ， Seq などの RDF コンテナや RDF コレクションといった機能や，識別子を持たない匿名の空白ノードなどがある．これらへ対応することは RDF を扱う上で重要な要素であるため今後の課題としたい．また，継承に関しても完全に対応し切れていない部分もある．たとえば，プロパティで定義する値域クラスと定義域クラスはサブプロパティへと継承される点や，文の述語として用いたプロパティが他のプロパティを継承している場合，スーパープロパティによっても問合せできるべきである点などである．これらの継承に関する点も今後考察する必要がある．

第 6 章 おわりに

最後に，Semantic Web 実現のためには，RDF 層 や RDF Schema 層より上の層のデータに対する処理手法も将来，必要になってくると考えている．本論文で対象としている RDF データは， Semantic Web のレイヤーケーキの RDF 層と RDF Schema 層の部分である．提案した手法もその基本概念に沿っており， RDF 層のインスタンスデータと RDF Schema 層のスキーマデータを区別して扱っている．RDF は Semantic Web 実現のために提案されたメタデータ記述のための基礎を提供しただけで，レイヤーケーキの一部でしかない．そのため，Semantic Web 実現のためには Ontology 層以上の層に関しても，大量のデータを扱うための手法が将来必要になってくる．

謝辞

本研究を行うにあたり，適切な御指導ならびにきめ細かな御配慮を賜わりました植村 俊亮教授に心から深く感謝の意を表し，御礼を申し上げます．植村先生には，本研究の主指導教官になっていただき，多数のコメントをいただきました．また，研究を遂行することのできる環境を提供していただきました．さらに，現在までに行ってきた研究発表について，研究指導としてのサポートだけではなく国内や海外における研究発表に関する旅費等の費用のサポートなど数多くの支援を賜りました．

本研究の指導教官になっていただきました関 浩之教授に深く御礼申し上げます．関教授が主催される XML 関連の研究会に幾度か参加させて頂きました．その際にも非常に丁寧なコメントを頂いたことや，発表の際に御意見を頂いたことが，本論文の基礎となりました．

本研究に対して有益な御助言，御協力を頂きました宮崎 純助教授に深く御礼申し上げます．宮崎先生には，本研究の指導教官になっていただき，親身になって非常に有益なご意見をいただきました．また，研究室で行われる全体研究会等においても有益なコメントを頂きました．頂いた多くの御意見は，本研究のみなく今後の研究においても参考になると考えております．

本研究の委員になっていただきました松本 祐治教授に深く御礼申し上げます．本論文に非常に有用なコメントやご意見を頂きました．本論文の執筆において不可欠なご意見であったと考えております．

本研究を遂行するにあたり，御指導を賜りまたした名古屋大学（前・奈良先端科学技術大学院大学 助教授）の吉川正俊教授に深く御礼申し上げます．吉川教授には，研究ミーティング等において，問題の本質を捉えた多くの御意見を頂きました．また，対外発表の際にも非常に参考になるコメントを数多く頂きました．

謝 辞

これらは本研究の成果を得るためには不可欠な御意見であったと考えております．また，研究を進める上での姿勢を学ぶことができました．今後の研究を遂行する基礎としたいと考えております．

本研究に対して有益な御助言，御協力を頂きました天笠 俊之助手に深く御礼申し上げます．本研究を遂行するために公私共にお世話になりました．本研究を含めた私の研究のミーティング，論文に対する指導において，親身になって私のことを考えていただいていることが非常に伝わっておりました．また，研究以外の面に関しても日ごろより御相談をさせていただき，大変お世話になりました．私が将来指導する立場になったときには，天笠先生にして頂いたように，親身になって指導していきたいと考えております．

本研究に対して有益な御助言，御協力を頂きました波多野 賢治助手に深く御礼申し上げます．本研究を遂行する上で，研究室の全体研究会等において貴重な御意見を賜り，また今後の方向性を決める上で多くの御助言をいただきました．研究以外でも，大変親しく接して頂き，公私においてさまざまなサポートをしていただきました．

学部生および博士前期課程に在籍していた岡山県立大学の横田教授および，言語・ソフトウェア工学研究室，知能メディア工学研究室の皆様に感謝いたします．私が岡山県立大学で学んだことは，本論文の一つの基礎になっており，岡山県立大学で培った研究に対する姿勢が本研究を遂行するための一つの原動力であったと考えております．

最後に，本研究に対する有益な御助言および御協力をいただいた奈良先端科学技術大学院大学情報科学研究科マルチメディア統合システム講座，奈良先端科学技術大学院大学情報科学研究科バイオ情報学領域データベース学分野研究室の学生，秘書の皆様に感謝いたします．私が本研究室に在籍していた 3 年間のあいだ，様々な方々から協力を受けることができました．研究室の皆様との人間関係の良さは私の研究の，また生活の原動力となったと思っています．

参考文献

[1] Rakesh Agrawal, Alexander Borgida, and H. V. Jagadish. Efficient Management of Transitive Relationships in Large Data and Knowledge Bases. In James Clifford, Bruce G. Lindsay, and David Maier, editors, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pp. 253–262. ACM Press, May 31 - June 2 1989.

[2] Sofia Alexaki, Vassilis Christophides, Greg Karvounarakis, Dimitris Plexousakis, and Karsten Tolle. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *Proceedings of the Second International Workshop on the Semantic Web (SemWeb'2001)*, pp. 1–13, May 1 2000.

[3] Dave J. Beckett. The design and implementation of the redland RDF application framework. In *Proceedings of the Tenth International World Wide Web Conference (WWW10)*, pp. 449–456, 2001.

[4] Elisa Bertino. Index Configuration in Object-Oriented Databases. *VLDB Journal*, Vol. 3, No. 3, pp. 355–399, 1994.

[5] Dan Brickley. Wordnet for the Web. `http://xmlns.com/2001/08/wordnet/`, August 2001.

[6] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proceedings of the First Internation Semantic Web Conference*, No. 2342 in Lecture Notes in Computer Science, pp. 54–68. Springer Verlag, July 2002.

[7] Wang chien Lee and Dik Lun Lee. Path Dictionary: A New Approach to Query Processing in Object-Oriented Databases, 1995.

[8] Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, and Sotirios Tourtounis. On labeling schemes for the semantic web. In *Proceedings of the twelfth international conference on World Wide Web*, pp. 544–555. ACM Press, 2003.

[9] Brian Cooper, Neal Sample, Michael J. Franklin, Gísli R. Hjaltason, and Moshe Shadmon. A Fast Index for Semistructured Data. In *The VLDB Conference*, pp. 341–350, 2001.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein. *Introduction to Algorithms*. The MIT Press, 2001. Second Edition.

[11] Dan Brickley and Libby Miller. FOAF Vocabulary Specification. http://xmlns.com/foaf/0.1/, 2003. RDFWeb Namespace Document 16 August 2003.

[12] Dave Beckett and Art Barstow. N-Triples. http://www.w3.org/2001/sw/RDFCore/ntriples/, September 2001. W3C RDF Core WG Internal Working Draft V1.9.

[13] Paul F. Dietz. Maintaining order in a linked list. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pp. 122–127. ACM Press, 1982.

[14] Dublin Core Metadata Element Set, Version 1.1. http://dublincore.org/documents/dces/, June 2 2003.

[15] GENE ONTOLOGY CONSORTIUM. Gene Ontology. http://www.geneontology.org/.

[16] Hewlett-Packard Company. RDQL – RDF Data Query Language. http://www.hpl.hp.com/semweb/rdql.htm.

[17] 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*, pp. 592–603. ACM Press, 2002.

[18] Raghav Kaushik, Pradeep Shenoy, Philip Bohannon, and Ehud Gudes. Exploiting Local Similarity for Efficient Indexing of Paths in Graph Structured Data. In *ICDE*, 2002.

[19] Donald E. Knuth. *The Art of Computer Programming*, Vol. 3, chapter Sorting and Searching. Addison-Wesley Publishing Company, 1998. Second Edition.

[20] Marja-Riitta Koivunen. W3C Semantic Web Activity. http://www.w3.org/Talks/2001/1102-semweb-fin/, November 2001.

[21] Quanzhong Li and Bongki Moon. Indexing and Querying XML Data for Regular Path Expressions. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pp. 361–370. Morgan Kaufmann Publishers Inc., 2001.

[22] Jason McHugh, Jennifer Widom, Serge Abiteboul, Qingshan Luo, and Anand Rajamaran. Indexing Semistructured Data. Technical report, Stanford University, Computer Science Department., 1998.

[23] Members of the TopicMaps.Org Authoring Group. Xml topic maps (xtm) 1.0. http://www.topicmaps.org/xtm/1.0/xtm1-20010806.html, August 2001.

[24] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. Introduction to WordNet: An On-Line Lexical Database. http://www.cogsci.princeton.edu/~wn/, 1993.

[25] Libby Miller. Inkling: RDF query using SquishQL. http://swordfish.rdfweb.org/rdfquery.

[26] Libby Miller, Andy Seaborne, and Alberto Reggiori. Three Implementations of SquishQL, a Simple RDF Query Language. In *Proceedings of the first Internatinal Semantic Web Conference (ISWC2002)*, 2002.

[27] Tova Milo and Dan Suciu. Index Structures for Path Expressions. In Catriel Beeri and Peter Buneman, editors, *Database Theory - ICDT '99, 7th International Conference*, Vol. 1540 of *Lecture Notes in Computer Science*, pp. 277–295. Springer, January 10-12 1999.

[28] Netscape. Open Directory Project. http://dmoz.org/.

[29] Online Computer Library Center. Dewey Decimal Classification. http://www.oclc.org/dewey/.

[30] Chen Qun, Andrew Lim, and Kian Win Ong. D(k)-Index: An Adaptive Structural Summary for Graph-Structured Data. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD2003)*, pp. 134–144, June 9-12 2003.

[31] Alberto Reggiori. RDFStore: Perl API for RDF Storage. http://rdfstore.sourceforge.net/.

[32] RSS-DEV Working Group. RDF Site Summary (RSS) 1.0. http://web.resource.org/rss/1.0/, December 2000.

[33] R.V. Guha. rdfDB : An RDF Database. http://guha.com/rdfdb/.

[34] Ron Sacks-Davis, Tuong Dao, James A. Thom, and Justin Zobel. Indexing Documents for Queries on Structure, Content and Attributes. In *Proceedings of the International Symposium on Digital Media Information Base*, pp. 236–245, 1997.

[35] Tim Berners-Lee. Notation 3: Ideas about Web Architecture - yet another notation. http://www.w3.org/DesignIssues/Notation3, November 2001. v.1.49.

[36] Kevin Wilkinson, Craig Sayers, Harumi A. Kuno, and Dave Reynolds. Efficient RDF Storage and Retrieval in Jena2. In *Proceedings of SWDB'03, The first International Workshop on Semantic Web and Databases, Berlin, Germany, September 7-8, 2003*, pp. 131–150, September 7-8 2003.

[37] Niklaus Wirth. Type Extensions. *ACM Transactions on Programming Languages and Systems*, Vol. 10, No. 2, pp. 204–214, apr 1988.

[38] World Wide Web Consortium. Xml path language (xpath) version 1.0. http://www.w3.org/TR/1999/REC-xpath-19991116. W3C Recommendation 16 November 1999.

[39] World Wide Web Consortium. XML Schema Part 2: Datatypes Second Edition. http://www.w3.org/TR/xmlschema-2/. W3C Recommendation 28 October 2004.

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

[41] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Third Edition). http://www.w3.org/TR/REC-xml, 2004. W3C Recommendation 04 February 2004.

[42] World Wide Web Consortium. RDF Primer. http://www.w3.org/TR/rdf-primer/, 2004. W3C Recommendation 10 February 2004.

[43] World Wide Web Consortium. RDF Semantics. http://www.w3.org/TR/rdf-mt/, 2004. W3C Recommendation 10 February 2004.

[44] World Wide Web Consortium. RDF Test Cases. http://www.w3.org/TR/rdf-testcases/, 2004. W3C Recommendation 10 February 2004.

[45] World Wide Web Consortium. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/, 2004. W3C Recommendation 10 February 2004.

[46] World Wide Web Consortium. RDF/XML Syntax Specification (Revised). http://www.w3.org/TR/rdf-syntax-grammar/, 2004. W3C Recommendation 10 February 2004.

[47] World Wide Web Consortium. Resource Description Framework (RDF). http://www.w3.org/RDF/, 2004. W3C Recommendation 10 February 2004.

[48] World Wide Web Consortium. Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/TR/rdf-concepts/, 2004. W3C Recommendation 10 February 2004.

[49] Zhaohui Xie and Jiawei Han. Join Index Hierarchies for Supporting Efficient Navigations in Object-Oriented Databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases*, pp. 522–533. Morgan Kaufmann, September 12-15 1994.

[50] Yohei Yamamoto, Masatoshi Yoshikawa, and Shunsuke Umeura. On Indices for XML Documents with Namespaces. In *Conference Proceedings of Markup Technologies*, 1999.

[51] 的野晃整, 天笠俊之, 吉川正俊, 植村俊亮. 接尾辞配列に基づいた RDF データのための索引手法. 情報処理学会論文誌: データベース, Vol. 45, No. SIG(TOD21), pp. 50–62, 2004 年 3 月.

[52] 的野晃整, 天笠俊之, 吉川正俊, 植村俊亮. 経路式に基づく RDF データの関係データベースへの格納と検索. 電子情報通信学会論文誌, Vol. J88-D-I, No. 3, pp. 590–603, 2005 年 3 月.

付録

A 実験で用いた索引

実験で用いた索引

```
1  CREATE INDEX cls_clsName_idx   ON class    USING HASH (className);
2  CREATE INDEX cls_num_idx       ON class    USING BTREE (pre, post, depth);
3  CREATE INDEX prop_name_idx     ON property USING HASH (propertyName);
4  CREATE INDEX prop_num_idx      ON property USING BTREE (pre, post, depth);
5  CREATE INDEX prop_domain_idx ON property USING BTREE (domain);
6  CREATE INDEX prop_range_idx    ON property USING BTREE (range);
7  CREATE INDEX type_res_idx      ON type     USING HASH (resourceName);
8  CREATE INDEX type_cls_idx      ON type     USING HASH (className);
9  CREATE INDEX triple_subj_idx   ON triple   USING HASH (subject);
10 CREATE INDEX triple_pred_idx   ON triple   USING HASH (predicate);
11 CREATE INDEX triple_obj_idx    ON triple   USING HASH (object);
12 CREATE INDEX path_pathexp_idx ON path      USING HASH (pathexp);
13 CREATE INDEX path_pathid_idx   ON path     USING BTREE (pathid);
14 CREATE INDEX res_pathid_idx    ON resource USING BTREE (pathid);
15 CREATE INDEX res_name_idx      ON resource USING BTREE (resourceName);
```

研究業績

学術雑誌論文

1. 的野 晃整, 天笠 俊之, 吉川 正俊, 植村 俊亮: “経路式に基づく RDF データの関係データベースへの格納と検索”, 電子情報通信学会論文誌和文論文誌, Vol.J88-D-I, No.3, pp.590-603, 平成 17 年 3 月 .

2. 的野 晃整, 天笠 俊之, 吉川 正俊, 植村 俊亮: “接尾辞配列に基づいた RDF データのための索引手法”, 情報処理学会論文誌:データベース, 第 45 巻, No. SIG4 (TOD 21), pp.50-62, 平成 16 年 3 月.

国際会議議事録（査読あり）

1. Akiyoshi Matono, Toshiyuki Amagasa, Masatoshi Yoshikawa, and Shunsuke Uemura: “A Path-based Relational RDF Database”, *The Sixteenth Australasian Database Conference (ADC2005)*, Newcastle, Australia, January 31st - February 3rd 2005 (to appear)

2. Akiyoshi Matono, Toshiyuki Amagasa, Masatoshi Yoshikawa, Shunsuke Uemura: “An Efficient Pathway Search using an Indexing Scheme for RDF”, *The Fourteenth International Conference on Genome Informatics (GIW 2003)*, pp.374-375, Yokohama, Japan, December 14-17, 2003. (poster presentation).

3. Akiyoshi Matono, Toshiyuki Amagasa, Masatoshi Yoshikawa, and Shunsuke Uemura: “An Indexing Scheme for RDF and RDF Schema based on Suffix Arrays”, *First International Workshop on Semantic Web and Databases (SWDB)*

co-located with 29th International Conference on Very Large Data Bases (VLDB2003), pp.151-168, Berlin, Germany, September 7-8, 2003.

研究会発表

1. 的野 晃整, 天笠 俊之, 吉川 正俊, 植村 俊亮: “経路式に基づく RDF データの関係データベースへの格納と検索”, 電子情報通信学会第 15 回データ工学ワークショップ (DEWS2004), 3-A-3, 2004 年 3 月 4-6 日, 伊勢志摩.

2. 的野 晃整, 天笠 俊之, 吉川 正俊, 植村 俊亮: “RDF のための経路式に基づいた索引方式”, 電子情報通信学会第 14 回データ工学ワークショップ (DEWS2003), 8-A-2, 2003 年 3 月 3-5 日, 加賀.

その他

レター

1. 的野 晃整, 天笠 俊之, 吉川 正俊, 植村 俊亮: “経路式に基づく RDF データの関係データベースへの格納と検索”, 日本データベース学会 Letters, Vol.3, No.1, pp.21-24, 2004 年 6 月.

全国大会口頭発表

1. 的野 晃整, 天笠 俊之, 吉川 正俊, 植村 俊亮: “経路式に基づく RDF データのための索引手法とその構築法”, 第 3 回情報科学技術フォーラム (FIT2004), 一般講演論文集第 2 分冊 D-037 pp.85-86, 2004 年 9 月 7 日-9 日, 京都府京田辺市.

その他の研究業績

著書（分担執筆）

1. Kazumasa Yokota, Takeo Kunishima, Akiyoshi Matono, and Bojiang Liu: “Management and Multimedia Presentation of Heterogeneous Documents” in *Non-traditional Database Systems*: Yahiko Kambayashi, Masaru Kitsuregawa, Akifumi Makinouchi, Shunsuke Uemura, Katsumi Tanaka, and Yoshifumi Masunaga (eds.), *Chapter 10*, The Information Processing Society of Japan and Taylor & Francis Ltd., ISBN 0-415-30206-4, 2002.

国内発表

1. 中尾伸章，天笠俊之，的野晃整，植村俊亮: “アクセス頻度を考慮した XML 文書分割方式の提案”, 電子情報通信学会 第 16 回データ工学ワークショップ (DEWS2005), 5-A-i5, 2005 年 3 月，佐世保.

2. 的野晃整, 野宮一生, 板谷昌洋, 国島丈生, 横田一正: “意味的に拡張した XML とその応用のためのデータベースエンジンの実現”, 電子情報通信学会第 13 回データ工学ワークショップ (DEWS2002), A5-2, 2002 年 3 月 4-6 日, 倉敷

3. 的野晃整, 藤野猛士, 平野尚孝, 板谷昌洋, 横田一正: “対話的演出機能をもつマルチメディア提示システムの実現”, 夏のデータベースワークショップ 2001 (DBWS2001), 情報処理学会データベースシステム, 電子情報通信学会データ工学合同研究会研究報告, Vol.2001, No.71, 2001-DBS-125 (II)-91, pp.201-208 / Vol.101, No.193, DE2001-104, pp.199-206, 2001 年 7 月 19 日. 函館

4. 的野晃整, 板谷昌洋, 横田一正, 國島丈生, 劉勃江: “データベース概念を組み込んだ XML のための問合せ言語”, 電子情報通信学会技術研究報告, Vol.101, No.110, DE2001-13, pp.97-104, 2001 年 6 月 12 日. 奈良

5. 的野晃整, 練苧宣行, 國島丈生, 横田一正: “マルチメディア情報を制御する QUIK のメディア管理”, 電気・情報関連学会中国支部第 51 回連合大会, 2000

年 10 月 21 日. 岡山大学.

6. 練苧宣行, 的野晃整, 國島丈生, 横田一正, “マルチメディア情報制御言語の実現”, 情報処理学会第 61 回全国大会, 第 3 分冊 pp.125-126, 2000 年 10 月 3-5 日. 愛媛大学.

7. 杉本健二, 緒方啓孝, 的野晃整, 平野尚孝, 横田一正, 國島丈生, “対話的マルチメディア情報提示システム実現のための QUIK の拡張”, 情報処理学会データベースシステム, 情報学基礎合同研究会研究報告, 2000-DBS-121-22 / 2000-FI-58-22, pp.153-160, 2000 年 5 月 25-26 日, 東京.