Linked Dataクエリ構築支援のための日本語文に基づくグラフ構造の生成

Linked Data クエリ構築支援のための日本語文に基づくグラフ構造の生成

筑波大学
図書館情報メディア研究科
2017 年 3 月
YAN JIANAN

目次

1. はじめに 1

2. 背景 2

2.1 Linked Data とその応用システムの開発 2
2.1.1 Linked Data と検索用クエリ言語 SPARQL 2
2.1.2 Linked Data 応用システムの開発 3
2.2 応用システムの開発における問題 4
2.2.1 Linked Data のデータセットの検索に関する問題 4
2.2.2 Linked Data クエリの構築 5
2.3 関連研究 8

3. クエリ構築支援のための日本語文に基づくグラフ構造の生成 10

3.1 クエリ構築の支援 10
3.2 係り受け解析器 Cabocha の構文解析結果に基づくグラフの生成 10
3.3 グラフ生成ルール 11
3.3.1 グラフ生成ルールの概要 11
3.3.2 文節結合ルール 11
3.3.3 トリプル生成ルール 13
3.4 グラフ生成処理の実現 14

4. 実験と結果 20

4.1 実験方針と対象データ 20
4.2 正解グラフの生成 20
4.3 実験結果 23

5. 考察 28

6. おわりに 30

謝辞 31

参考文献 32

1. はじめに

近年，Linked Data のデータセットをいくつか集め，それらを組み合わせて応用システムを作ることが増えてきた．Linked Data 応用システムの開発者が Linked Data のデータセットから欲しい情報を探すには，Linked Data のスキーマを確認しながら SPARQL クエリを組み立て検索する．しかしながら，スキーマ定義が記述されていない Linked Data が多いため，開発者はそのデータセットに対して様々なパターンの検索を試行錯誤し，スキーマを推定しながらクエリを組み立てる．それは手間がかかるため，本研究ではその検索の手間を減らすことを目指す手法を提案する．

本研究の目的は，Linked Data の構造を指定する検索クエリの書き方を知っている開発者向けに，日本語文からスキーマ定義が不明な Linked Data のデータセットを検索するためのクエリのグラフ構造を生成し，クエリの組み立てを容易にすることである．本研究では Linked Data のデータセットに RDF モデルを，クエリ言語に SPARQL を使用しているものを対象とする．

本論文では，まず第 2 章で Linked Data における背景知識とそのデータセットの検索に関する問題について述べる．第 3 章ではクエリ構築支援のために日本語文に基づいてグラフ構造を生成という考え方を説明し，それを実現するための提案手法について詳しく述べる．第 4 章と第 5 章では日本語文に基づくグラフ構造の生成手法を実際に適用し，そのグラフ生成能力を評価し，考察を行う．

2. 背景

2.1 Linked Data とその応用システムの開発

2.1.1 Linked Data と検索用クエリ言語 SPARQL

Linked Data はウェブ技術でデータを公開・共有するためのベストプラクティスであり，これまでウェブ上でデータ公開や共有について議論されてきたことの集大成である[1]. Linked Data のデータモデルとしては一般的に RDF が用いられている．RDF では，主語(Subject)・述語(Predicate)・目的語(Object) の組み合わせが最小単位となり，トリプルと呼ぶ．図 1 は，rdfs:label は目的語が主語のラベルであることを表す述語で，主語の IRI によって示されるものが「エンカナン」というラベルを持っていることを表している．IRI とは，URI 上での漢字がパーセントエンコーディングされてしまう問題を解決するため，Unicode 文字を直接扱えるように URI を拡張した識別子である[2].

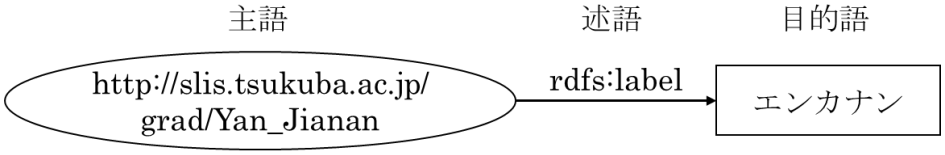

図 1　RDF データ例

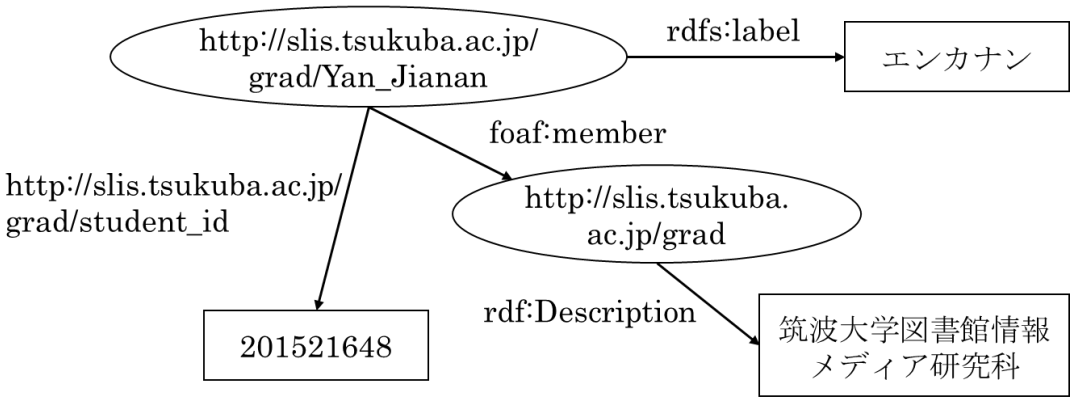

図 2　データセット中における複数のトリプルによる表現例

実際のデータセットは図 1 の例のような基本的なトリプルによって表された様々なデータを組み合わせた図 2 のようなものを集めたものである．そこから欲しい情報を探すには，検索クエリを組み立てる際に，主語・述語・目的語とその繋がり方を指定する必要がある．そのようなデータを検索するには，このようなトリプルの組み合わせによって表されるグラフ構造に対する問い合わせを記述する. RDF で表されたデータに対する問合せ言語で一般的に用いられているものの一つに SPARQL[3]があり，本研究でも SPARQL を対象とする．図 3 は図 2 のデータセットを対象に，名前と ID が目的語であることと，各々と繋がる主語と述語を事前に知り，ID が 201600000 より小さい学生の名前を求めようとするために書いた SPARQL クエリである．トリプル形式に従って，名前と ID の変数をその意味によって

「?name」と「?id」として，?name と繋がる主語「student:Yan_Jianan」と述語「rdfs:label」を指定し，?id と繋がる主語「student:Yan_Jianan」と述語「student:student_id」を指定してクエリを作る．図 3 中の 1 行目と 2 行目は，IRI を短縮して書くための接頭辞の宣言である．例えば，「@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>」というのは，以下の問合せ記述では，「<http://www.w3.org/2000/01/rdf-schema#>」を「rdfs」と表記するということである．図 3 中の 4 行目と 5 行目の「:」の後ろの部分は，省略された IRI の末尾である．例えば，「student:Yan_Jianan」は「http://slis.tsukuba.ac.jp/grad/Yan_Jianan」と同じ意味になる．また，6 行目と 7 行目は，?name がリテラルであることと，?id の値が 201600000 より小さいことを条件として示している．その実行結果の例を表 1 に示す．

```
1  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2  @prefix student: <http://slis.tsukuba.ac.jp/grad/>
3  select ?name where
4  {student:Yan_Jianan rdfs:label ?name;
5                              student:student_id ?id.
6  FILTER isLiteral(?name)
7  FILTER (?id < 201600000)
8  }
```

図 3　SPARQL クエリ例[†1]

表 1　クエリの実行結果例

name
エンカナン

例で示された以外にも，SPARQL クエリに指定できる要素が複数ある．例えば，変数の値に日付の範囲を指定したい場合，「FILTER(?変数名<= "YYYY-MM-DD"^^xsd:dateTime)」のように書く．その中で，「YYYY」には年を，MM には月を，DD には日を指定する．また，変数の値にある文字列が含まれると指定したい場合，「FILTER regex(?変数名,"含まれる文字列","i")」のように書く．「regex」は正規表現によるパターンマッチを意味し，「i」は変数の値に含まれる文字列が大文字と小文字に区別しないという意味を表している．それ以外に，膨大な結果に対して，ソリューション修飾子「GROUP BY」と「ORDER BY」を使うと，変数の値を集計してある順で並べることもできる．

2.1.2　Linked Data 応用システムの開発

Linked Data 応用システムは，開発者がいくつかのデータセットを組み合わせて作り上げたシステムのことである．例えば，「where does my money go?」[4]はイギリス政府によって公開された公共支出統計データを用いて開発されたシステムで，これをベースにして開発された日本語版もある．また，「さばえぶらり」[5]という鯖江市関連のデータセットと他 7

[†1] 図中の左側の番号は説明用の行番号であり，実際の記述に行番号はない．

種類の地図画像データが用いられて開発された地図システムもある．

そのようなシステムを開発するには，まず Linked Data の公開サイトやそこからデータを集めているデータカタログサイトを利用し，欲しいデータセットをダウンロードする．ダウンロードされたデータセットを自らの RDF ストアに入れて，SPARQL クエリを用いて欲しいデータを検索する． 一方，RDF ストアを用意して，SPARQL Endpoint という SPARQL クエリで検索できる Web API サービスを提供し，リアルタイム検索ができる Linked Data 公開サイトも存在しているが，その検索機能が提供されている RDF ストアのサーバに依存し, 取得できるデータ量も制限されていることも多い. そのため, システム構築者は自ら RDF ストアを準備することが多い.

2.2 応用システムの開発における問題

2.2.1 Linked Data のデータセットの検索に関する問題

前節で述べたように Linked Data 応用システムを開発するためには Linked Data データセットのスキーマを知る必要がある．しかし，実際の Linked Data データセットには，スキーマ定義が与えられない場合が多く存在している．西出らは，地方行政のデータセットを公開している CKAN 日本語[†2]と, 一般人からオープンデータを募集してコンテスト形式で評価し合う LOD チャレンジ Japan[6]のデータセットに対して調査した結果, 66 データセット中スキーマが公開されているのは 4 データセットだけであったと報告している[7].

スキーマ定義が明示的に与えられていない場合，それを知るために，SPARQL クエリを用いて様々なパターンの検索を繰り返し，スキーマを推定しながらクエリを組み立てるという方法が多く使われている．例えば，LOD チャレンジ 2013 受賞作品の一つである「トイレ危険地帯」の改良版「〇〇危険地帯」[8]は，大阪市のオープンデータを利用している．そこで使用された SPARQL クエリの一部を図 5 に示す．大阪市のオープンデータのスキーマ定義は公開されていて[9]，その抜粋を図 4 に示す．[9] よりダウンロードした「osaka_shisetsu2_ttl.txt」というファイルの 58 行目まではスキーマ定義であり，以降はインスタンスデータである．このような定義があるので，図 5 のような SPARQL クエリが簡単に書けるが，もしそれが公開されていない場合，5 行目の「"公衆トイレ"@ja」を目的語として，それに繋がる述語を調べることから始めないと，図 5 のようなクエリを簡単に組み立てられないと考えられる．

そこで，クエリ組み立てのヒントを開発者に提示すれば，検索の手間も軽減されると思われる．スキーマ定義が不明であるため，情報要求からクエリの組み立てに役立つグラフ構造を生成し，それを開発者に提示することが考えられる．

[†2] CKAN 日本語は 2016 年 9 月 5 日に運用を終了しており，そのデータは現在 LinkData.org より提供されている．

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix cc:  <http://creativecommons.org/ns#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
<>
        cc:attributionName "nagai"@ja ;

        cc:license <http://creativecommons.org/licenses/by/3.0/deed.ja> .

<http://linkdata.org/property/rdf1s933i#category_1>
        rdf:type rdf:Property ;
        rdfs:label "category_1"@ja .
<http://linkdata.org/property/rdf1s933i#category_2>
        rdf:type rdf:Property ;
        rdfs:label "category_2"@ja .
<http://linkdata.org/property/rdf1s933i#ku>
        rdf:type rdf:Property ;
        rdfs:label "ku"@ja .
…
```

図 4　大阪市の施設情報のスキーマの抜粋

```
SELECT DISTINCT *
FROM
<http://lod.sfc.keio.ac.jp/challenge2013/show_status.php?id=d030>
WHERE{
   ?uri <http://lodosaka.hozo.jp/category_1> "公衆トイレ"@ja ;
        <http://www.w3.org/2003/01/geo/wgs84_pos#lat> ?latitude ;
        <http://www.w3.org/2003/01/geo/wgs84_pos#long> ?longitude .
}
```

図 5　大阪市のトイレ位置情報を取得する SPARQL クエリ

2.2.2　Linked Data クエリの構築

本研究では，開発者が日本語の情報要求を入力すると，それに基づいた検索クエリのグラフが生成されるシステムを目指す．その部分クエリを RDF ストアで検索し，提示された述語候補から選択する．全ての述語候補が確定すれば，クエリが完成される．全体の流れは以下のようになる：

- 開発者が情報要求を日本語文で入力すると，システムがその情報要求をクエリのグラフ

に変換し，出力する．

- 開発者がクエリのグラフに基づいて部分クエリを用いて検索を行い，提示された述語の候補とその出現頻度から選ぶ．
- 直前のステップを繰り返してクエリを完成する．
- 途中で適切な述語の候補を見つからない場合，開発者がクエリのグラフの別候補を選択するか，クエリの一部を修正する．

そこに必要な技術要素を明確にするために，クエリ構築の過程を人手で行った事例を示す．

Dbpedia[10] 2015-4 のデータセットを使用し，「防水性能を持つデバイスの機種を知りたい」を情報要求例として検索クエリ完成までの過程を追った．

設計したクエリ構築の過程を例で説明すると，以下の通りになる：

- 「防水性能を持つデバイスの機種を知りたい」という情報要求を文節に分割する（図 6）．

A	p1	X	Y, p2
（防水性能を）	（持つ）	（デバイスの）	（機種を）（知りたい）

図 6　情報要求の分割結果

- 図 6 の分割結果により，1 つの係り受け関係を 1 つの RDF トリプルとする．その係り受け関係を図 7 に示す．また，検索したい部分を主語とし，動詞を含む部分を述語とし，具体値を示す名詞や形容詞を含む部分を目的語とする．それらのトリプルの組み合わせが検索クエリのグラフとなる（図 8）．

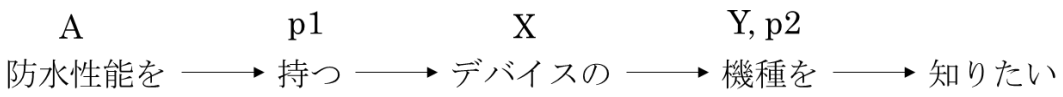

図 7　情報要求の係り受け関係

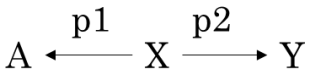

図 8　検索クエリのグラフ

- クエリのグラフに基づいて，X と A を p1 で繋ぐ部分グラフに対して，「防水性能」をリテラルで表すと，A の値に「waterproof」が含むと考えられるため，それを FILTER に指定して部分クエリ 1 で検索を行い，述語の候補 p1 とその出現頻度を提示する．部分クエリ 1 は図 9 のようになり，その検索結果を表 2 に示す．

```
select distinct ?p1, (count(?p1) as ?cp) from <http://dbpedia.org> ... where
{ ?X ?p1 ?A.
FILTER regex(?A,"waterproof","i")}
group by ?p1
order by DESC(?cp)
```

図 9　部分クエリ 1

表 2　部分クエリ 1 の検索結果

p1	cp
http://dbpedia.org/ontology/wikiPageWikiLink	459
http://dbpedia.org/ontology/abstract	444
http://dbpedia.org/ontology/wikiPageWikiLinkText	87
http://www.w3.org/2000/01/rdf-schema#label	36
http://dbpedia.org/property/shortsummary	18
http://dbpedia.org/ontology/wikiPageExternalLink	12
...	...

- 表 2 より紹介文の意味を表している「http://dbpedia.org/ontology/abstract」を選び，部分クエリ 1 を完成する．次の X と Y を p2 で繋ぐ部分グラフに対して，Y が文字列であるという条件を付けて，部分クエリ 1 と繋いで検索を行い，述語の候補 p2 とその出現頻度を提示する. 部分クエリ 2 は図 10 のようになり, その検索結果を表 3 に示す.

```
prefix a: <http://dbpedia.org/ontology/>
select distinct ?p2, (count(?p2) as ?cp) from <http://dbpedia.org> ... where
{ ?X a:abstract ?A;
      ?p2 ?Y.
FILTER regex(?A,"waterproof","i")
FILTER isLiteral(?Y)
}
group by ?p2
order by DESC(?cp)
```

図 10　部分クエリ 2

表 3　部分クエリ 2 の検索結果

p2	cp
http://dbpedia.org/ontology/wikiPageWikiLinkText	1557
http://www.w3.org/2000/01/rdf-schema#label	1274
http://dbpedia.org/ontology/abstract	1252
http://dbpedia.org/ontology/wikiPageOutDegree	444
http://dbpedia.org/ontology/wikiPageLength	444
http://dbpedia.org/property/name	127
...	...

- 表 3 より名前の意味を表している「http://dbpedia.org/property/name」を選び，部分クエリ 2 を完成すれば，クエリも完成する．完成されたクエリを図 11 に示し，最後の検索結果の一部を表 4 に示す．表 4 の「@en」は，言語の英語を表している．

```
prefix a: <http://dbpedia.org/ontology/>
prefix b: <http://dbpedia.org/property/>
select distinct ?Y
from <http://dbpedia.org> ...where
{ ?X a:abstract ?A;
      b:name ?Y.
FILTER regex(?A,"waterproof","i")
FILTER isLiteral(?Y)
}
```

図 11　完成されたクエリ

表 4　最後の検索結果（122 件）の一部

Y
“Sony Xperia ZR”@en
“PRB 408”@en
“Xperia Z”@en
“Sony Xperia Z1”@en
“Sony Xperia Z1 Compact”@en
“Fujitsu Toshiba IS12T”@en
…

2.3　関連研究

構造を指定するクエリを書く知識が欠けているユーザ向けに，キーワード指定クエリから段階的に詳細化していく検索支援手法 MorphingAssist[11]がある．MorphingAssist は，テキストページとグラフデータの組からなる Web データに対する問合せ言語 Gradation の支援ツールであり，ユーザがキーワードを入力すると，Gradation クエリを書くためのヒントが提示される．しかし，MorphingAssist は Gradation に基づいてクエリを書くためのヒントを提示するため，この手法はシソーラスと一部の図書館書誌データのようなテキストページと結び付いていないデータに対して，適用できないという問題がある．本研究では，データがテキストページと結び付いているか否かに関係なく，クエリの構築を支援する．

また，メタデータインスタンスと SPARQL クエリを用いたスキーマを抽出する Honma らの手法[12]がある．Honma らは，Linked Open Data のデータセットからスキーマを Dublin Core Description Set Profile (DSP) [13]として自動的に抽出できる手法を提案した．しかし，この手法は rdf:type に依存するため，それが用いられていないデータセットには適用できない．本研究では，データセットからスキーマを抽出ではなく，日本語の情報要求に基づいてグラフ構造を生成し，それをスキーマとしてクエリの組み立てを支援する．

SPARQL クエリの自動生成ツールとしては，SPARQL Builder[14]がある．SPARQL Builder では，入力と出力の 2 つのクラスを指定すれば，その二つのクラスの間のパスが計算され，クラス間関係リストがユーザに提示される．ユーザがそのリストから 1 つ選ぶと，

そのパスのクラス間関係に対応する SPARQL クエリが自動的に生成される．このツールの欠点としては，SPARQL Builder は生命科学分野の RDF データに限定されていて，スキーマ定義も既に準備されているため，スキーマ定義がない他の分野の Linked Data には適用できない．本研究では，Linked Data の分野を限定しなくても SPARQL クエリの作成を支援できる手法を提案する．

一方，リレーショナル・データベースのための対話型自然言語インタフェースの構築[15]は，ユーザが入力した質問文を木構造に変換し，木構造の各ノードを SQL クエリの構成要素に対応してクエリを生成する手法であるが，その構成要素に対応するには，メタデータが必要である．本研究では，Linked Data を対象として自然言語からグラフ構造に変換し，開発者がそれを参考して SPARQL クエリを組立てることを支援する．

3. クエリ構築支援のための日本語文に基づくグラフ構造の生成

3.1 クエリ構築の支援

本研究では，日本語文で書かれた情報要求から検索クエリのグラフを生成し，開発者がそのグラフに基づいて未知の述語を探し，クエリの作成を支援する．開発者はクエリの書き方を知っているほか，RDF データで一般的に用いられる RDFS や FOAF 等の語彙についても知っており，述語の候補を見ればある程度その述語の意味を理解できるものと仮定する．案した支援手法の流れは以下のようになる：

- 日本語文で書かれた情報要求を構文解析し，出力結果によって得られた情報を文節ごとに分割し，係り受け関係を得る．
- 各文節間の係り受け関係に基づいて，グラフ生成ルールを定める
- グラフ生成ルールに沿って必要の部分を取り出し，組み合わせて検索クエリのグラフを生成する．
- 開発者が生成された複数のグラフの候補から一つを選び，それによって述語を探し，部分ずつにクエリを完成する．

3.2 係り受け解析器 Cabocha の構文解析結果に基づくグラフの生成

クエリの構築を支援するため，日本語係り受け解析器 Cabocha[16]を利用し，その出力結果に基づいてクエリのグラフを自動的に生成する．2.2.2 で使用した「防水性能を持つデバイスの機種を知りたい」を情報要求例として用いて，Cabocha によって得られた情報を図 12 に示す．図 12 では，「*」で始まる行から次の「*」の前までは 1 つの文節を表し，「*」の右の数値は文節番号である．文節番号の後に続く数値は順に係り先の文節番号，主辞の形態素番号/機能語の形態素番号，係り関係のスコアとなる．係り先なしの場合は，係り先の文節番号は-1 となる．

```
防水性能を持つデバイスの機種を知りたい
* 0 1D 1/2 2.061194
防水    名詞,サ変接続,*,*,*,*,防水,ボウスイ,ボースイ,,
性能    名詞,一般,*,*,*,*,性能,セイノウ,セイノー,,
を      助詞,格助詞,一般,*,*,*,を,ヲ,ヲ,,
* 1 2D 0/0 0.810675
持つ    動詞,自立,*,*,五段・タ行,基本形,持つ,モツ,モツ,もつ/持つ,
* 2 3D 0/1 1.912269
デバイス        名詞,一般,*,*,*,*,デバイス,デバイス,デバイス,,
の      助詞,連体化,*,*,*,*,の,ノ,ノ,,
* 3 4D 0/1 1.912269
機種    名詞,一般,*,*,*,*,機種,キシュ,キシュ,,
を      助詞,格助詞,一般,*,*,*,を,ヲ,ヲ,,
* 4 -1D 0/1 0.000000
知り    動詞,自立,*,*,五段・ラ行,連用形,知る,シリ,シリ,しり/知り,
たい    助動詞,*,*,*,特殊・タイ,基本形,たい,タイ,タイ,,
EOS
```

図 12　情報要求例の構文解析結果

図 12 のような Cabocha の解析結果をグラフに変換する準備をするため，まず分割された文節を格納するためのオブジェクトを用意する．文節を表すオブジェクトの構造を図 13 に示す．図 12 の情報要求例の構文解析結果を例として説明すると，phraseNum は「*」の後

の１番目の数値，文節番号を格納するためのフィールドであり，dependentNum は次の「D」を除く数値，係り先の文節番号を格納するためのフィールドであり，mainWordNum は次の「/」の前の数値，その文節の主辞を格納するためのフィールドであり，functionWordNum はその「/」の後の数値，その文節の機能語を格納するためのフィールドであり，wordsList は残りの部分，その文節に含まれる単語と品詞を格納するフィールドである．

```
// 文節を表すオブジェクトのクラス(実質構造体)
class Phrase {
    int phraseNum;          // 文節番号
    int dependentNum;       // 係り先の文節番号
    int mainWordNum;        // 主辞番号
    int functionWordNum;    // 機能語番号
    ArrayList wordsList;    // 単語と品詞を格納するリスト
}
```

図 13　文節を表すオブジェクトの構造

3.3　グラフ生成ルール

3.3.1　グラフ生成ルールの概要

本節では，構文解析ツールで情報要求を解析し，その結果に基づいてクエリのグラフを生成する規則について述べる．

日本語係り受け解析器 Cabocha を利用し，その出力結果をクエリのグラフに変換する規則を定め，自動変換を試みる．規則の検討においては，まず 2.2.2 で取り上げた情報要求例について変換できるものを定義する．そして，それを他の情報要求例にも適用し，不十分なところを補うことでより汎用性の高い規則とする．

まず，簡単な場合について規則が定義可能であることを確かめるため，次のような制約がある場合について考える．

- 情報要求は「～を知りたい」という形式に限定するものとする．
- 生成するグラフにおいて，RDF の主語は 1 つであると仮定する．
- 最後に得るのはリテラルである場合に限る．

変換ルールを文節結合ルールとトリプル生成ルールに分け，もし結合すべき文節があれば，それらに対して文節結合ルールを適用することで，細かく分かれ過ぎた文節を結合し，その後トリプル生成ルールを適用する．もし結合すべき文節がなければ，直接にトリプル生成ルールを適用する．

3.3.2　文節結合ルール

文節結合ルールを定義する理由としては，情報要求を Cabocha で解析すると，例えば図 14 で示すように，生年月日などは RDF データでは一つのノードとして表されるので，それらを一つの文節にする必要があるためである．文節結合ルールは図 16 のように定義した．以下，文節番号を i，節 i を C_i，主語 j を S_j，目的語 i を O_i，S_j と O_i を繋ぐ述語を P_{ji} として表す．ここで主語，述語，目的語は RDF モデルにおけるものである．図 14 の文節を結合した結果として，得たい係り受け関係を図 15 に示す．

```
1960年1月1日以前に生まれた芸術家の名前を知りたい
* 0 1D 4/4 1.482436
1       名詞,数,*,*,*,*,1,イチ,イチ,,
9       名詞,数,*,*,*,*,9,キュウ,キュー,,
6       名詞,数,*,*,*,*,6,ロク,ロク,,
0       名詞,数,*,*,*,*,0,ゼロ,ゼロ,,
年      名詞,接尾,助数詞,*,*,*,年,ネン,ネン,,
* 1 2D 0/0 0.740106
1月     名詞,副詞可能,*,*,*,*,1月,イチガツ,イチガツ,,
* 2 3D 2/3 1.416432
1       名詞,数,*,*,*,*,1,イチ,イチ,,
日      名詞,接尾,助数詞,*,*,*,日,ニチ,ニチ,,
以前    名詞,副詞可能,*,*,*,*,以前,イゼン,イゼン,,
に      助詞,格助詞,一般,*,*,*,に,ニ,ニ,,
* 3 4D 0/1 0.874276
生まれ  動詞,自立,*,*,一段,連用形,生まれる,ウマレ,ウマレ,うまれ/生まれ/生れ,
た      助動詞,*,*,*,特殊・タ,基本形,た,タ,タ,,
* 4 5D 1/2 1.609654
芸術    名詞,一般,*,*,*,*,芸術,ゲイジュツ,ゲイジュツ,,
家      名詞,接尾,一般,*,*,*,家,カ,カ,,
の      助詞,連体化,*,*,*,*,の,ノ,ノ,,
* 5 6D 0/1 1.609654
名前    名詞,一般,*,*,*,*,名前,ナマエ,ナマエ,,
を      助詞,格助詞,一般,*,*,*,を,ヲ,ヲ,,
* 6 -1D 0/1 0.000000
知り    動詞,自立,*,*,五段・ラ行,連用形,知る,シリ,シリ,しり/知り,
たい    助動詞,*,*,*,特殊・タイ,基本形,たい,タイ,タイ,,
EOS
```

図 14　文節結合が必要の情報要求例の構文解析結果

（1960年1月1日以前に）→（生まれた）→（芸術家の）→（名前を）→（知りたい）
C_1　　C_2　　C_3　　C_4

図 15　文節結合が必要の情報要求例の係り受け関係

if $C_i \rightarrow C_j$, $f(C_i) = N$, $m(C_i) = N$, $m(C_j) = N$
　then C_{out} is $C_i + C_j$, $m(C_{out})$ is $m(C_j)$, $f(C_{out})$ is $f(C_j)$

記号の説明：
　Ci　：文節 i
　f(Ci)：Ci の機能語
　m(Ci)：Ci の主辞
　N：名詞
　Cj → Ci：Cj が Ci に係る
　,：AND

図 16　文節結合ルール

図 16 では，もしある節 C_i の機能語と主辞が名詞であり，係る先の節 C_j の主辞が名詞であれば，C_i と C_j を 1 つの節 C_{out} と見なし，C_k の機能語と主辞を C_j と同じにすることを意味している．

3.3.3 トリプル生成ルール

結合後の文節の係り受け関係によって，RDF トリプル構造を生成する．トリプル生成ルールは図 17 のように定義した．

1. if $C_j \rightarrow C_i \rightarrow$ “知りたい”, $m(C_i) = N$, $f(C_j) = Aux$
   then S_j is $m(C_j)$, O_i is $m(C_i)$, Triple is $\{S_j, P_{ji}, O_i\}$
2. if $C_l \rightarrow C_k \rightarrow C_j \rightarrow$ “知りたい”, $m(C_k) = V$
   then P_k is $m(C_k)$, S_l is $m(C_l)$, O_j is $m(C_j)$, Triple is $\{S_l, P_k, O_j\}$
3. if $C_l \rightarrow C_k \rightarrow C_j$ ↛ “知りたい”, $m(C_k) = V$
   then P_k is $m(C_k)$, S_j is $m(C_j)$, O_l is $m(C_l)$, Triple is $\{S_j, P_k, O_l\}$
4. If $C_k \rightarrow C_j$, $m(C_k) = Adj$
   then O_k is $m(C_k)$, Triple is $\{S_j, P_{jk}, O_k\}$
5. If $C_k \rightarrow C_j$, $m(C_j)$ is S_j , $m(C_k) = N$
   then O_k is $m(C_k)$, Triple is $\{S_j, P_{jk}, O_k\}$

記号の説明：

O_i：目的語
S_j：主語
P_{ji} または P_k：述語
Aux：助詞
Adj：形容詞
V：動詞
Cj ↛ Ci：Cj が Ci に係らない
Triple is $\{S_j, P_{ji}, O_i\}$：トリプル

図 17 トリプル生成ルール

図 17 のルール 1 から 5 の詳しい説明は以下のようになる：

1. もし節 C_j が節 C_i に係って，節 C_i が「知りたい」に係って，節 C_i の主辞が名詞であり，且つ節 C_j の機能語が助詞であれば，節 C_j の主辞が主語に相当し，これを S_j とし，節 C_i の主辞が目的語に相当し，これを O_i とし，S_j と O_i を繋ぐ述語を P_{ji} としたトリプル $\{S_j, P_{ji}, O_i\}$とする．
2. もし節 C_l が節 C_k に係って，節 C_k が節 C_j に係って，節 C_j が「知りたい」に係って，且つ節 C_k の主辞が動詞であれば，節 C_k の主辞を S_l と O_j を繋ぐ述語 P_k としたトリプル$\{S_l, P_k, O_j\}$とする．
3. もし節 C_l が節 C_k に係って，節 C_k が節 C_j に係って，節 C_j が「知りたい」に係っていなく，且つ節 C_k の主辞が動詞であれば，節 C_k の主辞を S_j と O_l を繋ぐ述語 P_k としたトリプル$\{S_j, P_k, O_l\}$とする．
4. もし節 C_k が節 C_j に係って，且つ節 C_k の主辞が形容詞であれば，節 C_j の主辞が主語に相当し，これを S_j とし，節 C_k の主辞が目的語に相当し，これを O_k とし，S_j と O_k を繋ぐ述語を P_{ji} としたトリプル$\{S_j, P_{jk}, O_k\}$とする．
5. もし節 C_k が節 C_j に係って，節 C_j の主辞が主語であり，且つ節 C_k の主辞が名詞であれば，節 C_j の主辞を S_j とし，節 C_k の主辞が目的語に相当し，これを O_k とし，S_j と O_k を繋ぐ述語を P_{ji} としたトリプル$\{S_j, P_{jk}, O_k\}$とする．

これらのルールに沿ってトリプルの集合が生成されるので, それに基づいて SPARQL クエリの条件を生成する．その際，そのクエリ中に含まれるノードのうち，名詞である主辞に対応するものについては，その名詞の意味を表す言葉を条件として FILTER 句として追加してもらう．

3.4 グラフ生成処理の実現

3.2 で述べた分割された文節を格納するためのオブジェクトを用意した後, それをグラフに変換するためのメソッドを定義する．その処理の擬似コードを図 18 に示す．前章で述べた文節結合ルールとトリプル生成ルールを実現するための処理を以下に示す．

グラフ生成処理の擬似コードを図 19 に示す．グラフ生成処理の中の文節結合ルールの処理の擬似コードを図 20 に示す．文節結合処理を行った後，トリプル生成ルールの処理を行い，その擬似コードを図 21 から図 25 に示す．

```
// 変換処理をするためのメソッドを定義するためのクラス(インスタンスなし)
public class RunCabocha {
// 変換処理メソッド
    public void Run() {
        try {
            必要なパラメータを渡して cabocha を実行開始，出力を読み込むための
            オブジェクトを br に入れる．
            文節オブジェクトを格納する phraseList を用意し，空にする．
            一時変数 phrase を宣言し，初期値を空とする．
            while ((line = br.readLine()) != null) {
              if (line が「EOS」があれば) {
                if (phrase != null) {
                    phrase の値を phraseList の最後に追加する
                }
                break;          // 読み込み処理を終了
              }
              if (line の先頭に「*」があれば) {
                  // 文節情報の 1 行目の処理
                if (phrase != null) {
                    phrase の値を phraseList の最後に追加する
                }
                phrase = new Phrase();
                line の内容を分割して文節番号等を phrase の対応するフィールド
                に入れる．
                phrase の wordsList を空にする
              } else {
                    // 文節に含まれている形態素(単語)の処理
                    line の内容を「¥t」或いは「,」で分割して文字列の配列とし，
                    その配列を phrase の wordsList に追加する．
                    }
            }
        phraseList に入っている文節オブジェクトを用いてグラフを生成する（グラ
        フ生成処理)
        } catch (例外型の指定) {
            例外時の処理
         }
    }
    public static void main (String[] args) {
        RunCabocha run = new RunCabocha();
        run.Run();
    }
}
```

図 18　文節をグラフに変換するためのメソッドを定義する処理の擬似コード

```
// phraseList に入っている文節オブジェクトを用いてグラフ生成する
for (int n = 0; n < phraseList.size(); n++) {
        phraseList の先頭から順に取り出し, 1 番目の文節を phrase1 に入れる.
        phrase1 の係り先の文節があれば，その文節を phrase2 に入れる.
        （phrase2 から phrase5 まで同上）
        if(phrase5 の dependentNum == -1){
        }
        phrase1 の主辞と機能語を取り出し，それを格納するための配列
        mainWordArray1 と functionWordArray1 を用意する.
        （phrase2 から phrase5 まで同上）
        トリプルを格納するリスト triple を用意する.
        triple を格納するリスト tripleList を用意する.
        主語，述語，目的語の String 型変数 s，p，o を用意する.
        文節結合ルールの処理
        トリプル生成ルール 1 の処理
        トリプル生成ルール 2 の処理
        トリプル生成ルール 3 の処理
        トリプル生成ルール 4 の処理
        トリプル生成ルール 5 の処理
        tripleList の内容を出力する
}
```

図 19　グラフ生成処理の擬似コード

```
// もし結合すべき文節があれば，以下の文節結合ルールの処理をしてからトリプル生成ルールの処理をする
if (functionWordArray1[1]が「名詞」であり &&
    mainWordArray1[1]が「名詞」であり &&
    mainWordArray2[1] が「名詞」であれば
    ) {
      int wordsListSize = phrase1.wordsList.size();
      結合後の文節を格納するためのリスト newWordsList を用意する
      for(int i = 0; i < phrase1.wordsList.size(); i++){
         phrase1 の wordsList を newWordsList に入れる
                                                    }
      for(int j = 0; j < phrase2.wordsList.size(); j++){
         phrase2 の wordsList を newWordsList に入れる
                                                    }
      phrase1 と phrase2 の wordsList を空とする.
      for(int k = 0; k < newWordsList.size(); k++){
         newWordsList を phrase2 の wordsList に入れる
                                              }
      phrase2 の functionWordNum += wordsListSize
      phrase2 の mainWordNum += wordsListSize;
            }
```

図 20　文節結合ルールの処理の擬似コード

```
//「知りたい」に係る目的語とそれに繋がる主語を指定する条件分岐（トリプル生成ルール 1）
if(phrase5 の dependentNum == -1 &&
   mainWordArray5[0]が「知り」であり &&
   functionWordArray5[0] が「たい」であり &&
   mainWordArray4[1] が「名詞」であり &&
   functionWordArray3[1] が「助詞」であれば)
   {
    s = mainWordArray3[0]
    p = "p" + 文字列に変換された phrase3 の phraseNum + 文字列に変換された
    phrase4 の phraseNum
    o = mainWordArray4[0]
    s,p,o を triple の 0 番から 2 番に入れる
    triple を tripleList に格納する
    }
```

図 21　トリプル生成ルール 1 の処理の擬似コード

```
// phrase5 が係り先なしの場合，目的語と主語をつなぐ述語を指定する条件分岐（トリプル生成ルール 2）
if(phrase5 の dependentNum == -1 && mainWordArray3[1] が「動詞」であれば){
        s = mainWordArray2[0]
        p = mainWordArray3[0]
        o = mainWordArray4[0]
        s,p,o を triple の 0 番から 2 番に入れる
        triple を tripleList に格納する
        }
```

図 22　トリプル生成ルール 2 の処理の擬似コード

```
// もし文節結合の処理を行わなかったら，phrase4 が係り先がある場合，目的語と主語をつなぐ述語を指定する条件分岐（トリプル生成ルール 3）
if(phrase4 の dependentNum != -1 && mainWordArray2[1] が「動詞」であれば){
        s = mainWordArray3[0]
        p = mainWordArray2[0]
        o = mainWordArray1[0]
        s,p,o を triple の 0 番から 2 番に入れる
        triple を tripleList に格納する
        }
// もし既に文節結合の処理を行ったら，phrase5 が係り先なしの場合，目的語と主語をつなぐ述語を指定する条件分岐（トリプル生成ルール 3）
if(phrase5 の dependentNum != -1 && mainWordArray3[1] が「動詞」であれば){
        s = mainWordArray4[0]
        p = mainWordArray3[0]
        o = mainWordArray2[0]
        s,p,o を triple の 0 番から 2 番に入れる
        triple を tripleList に格納する
        }
```

図 23　トリプル生成ルール 3 の処理の擬似コード

```
// 形容詞を目的語に指定する条件分岐（トリプル生成ルール 4）
if(mainWordArray1[1] が「形容詞」であれば){
   s = mainWordArray2[0]
   p = "p" + 文字列に変換された phrase2 の phraseNum + 文字列に変換さた
   phrase1 の phraseNum
   o = mainWordArray1[0]
   s,p,o を triple の 0 番から 2 番に入れる
   triple を tripleList に格納する
                                                    }
```

図 24　トリプル生成ルール 4 の処理の擬似コード

```
// 名詞を目的語に指定する条件分岐（トリプル生成ルール 5）
if(mainWordArray2[0]が主語であり  && mainWordArray1[1] が「名詞」で
あれば){
        s = mainWordArray2[0]
        p = "p" + 文字列に変換された phrase2 の phraseNum + 文字列に変換
        された phrase1 の phraseNum
        o = mainWordArray1[0]
        s,p,o を triple の 0 番から 2 番に入れる
        triple を tripleList に格納する
        }
```

図 25　トリプル生成ルール 5 の処理の擬似コード

4. 実験と結果

4.1 実験方針と対象データ

本研究で行った実験の環境と対象データは以下のようになる.

- RDF ストア：Virtuoso 07.20.3212
- OS：Ubuntu 14.04 LTS
- 対象データ：Dbpedia 2015-4（669,045,743 件）

以上の環境とデータを用いて，まず情報要求に基づいて人手でクエリを作成し，そのクエリのグラフと検索結果を本研究での正解として扱う．次に提案手法によるグラフを生成し，それに基づいてクエリを組み立て，検索結果を示す．生成したグラフと検索結果によって次の 2 つの観点で評価を行う.

(1) 正解グラフを比較対象として，それと提案手法によるグラフを比較し，一致するトリプル数はどれくらいあるか
(2) 提案手法によるグラフに基づく結果は正解のグラフに基づく結果と一致する比率はどれくらいあるか

4.2 正解グラフの生成

Dbpedia 2015-4 のデータセットを情報要求に対する検索対象として, それに対して検索結果が出ると考えられる日本語の情報要求 3 件を用いて，グラフ生成の評価実験を行った．その 3 件の日本語の情報要求を表 5 に示す.

表 5　評価実験で用いた日本語の情報要求

情報要求番号	情報要求文
1	防水性能を持つデバイスの機種を知りたい
2	漫画を原作とした映画のタイトルを知りたい
3	パソコンとスマートフォンを販売しているブランドを知りたい

まず, 正解を設定するため, グラフ生成ルールを使わずに人手で表 5 の情報要求で Dbpedia 2015-4 のデータセットを検索する．「防水性能を持つデバイスの機種を知りたい」を例として手順例と結果の一部を以下に示す.

- 私が知っている防水性能を持つデバイスの機種名「Sony Xperia ZR」をリテラルにして，目的語として対応する主語の IRI を探し，その最初のクエリを図 26 に示す.

```
select ?s from <http://dbpedia.org> ... where
{ ?s ?p1 “Sony_Xperia_ZR”@en.
}
```

図 26　最初のクエリ

- 末尾が「Sony_Xperia_ZR」の IRI「http://dbpedia.org/resource/Sony_Xperia_ZR」を選び，それを主語として検索し，検索結果から防水性能を持つということを表す述語と目的語を探す.
- 検索結果から説明文の意味を表す述語「http://dbpedia.org/ontology/abstract」を選び, 主語である「http://dbpedia.org/resource/Sony_Xperia_ZR」と繋いで部分クエリ A

となり，図 27 に示す.

```
prefix a: <http://dbpedia.org/ontology/>
select distinct ?deviceNameLiteral
from <http://dbpedia.org> ... where
{ ?deviceIRI a:abstract ?deviceAbstract.
}
```

図 27　部分クエリ A

- 検索結果から名前を表す述語「http://dbpedia.org/property/name」を選び，主語である「http://dbpedia.org/resource/Sony_Xperia_ZR」と繋いで部分クエリ B となり，図 28 に示す.

```
prefix a: <http://dbpedia.org/ontology/>
prefix b: <http://dbpedia.org/property/>
select distinct ?deviceNameLiteral
from <http://dbpedia.org> ... where
{ ?deviceIRI a:abstract ?deviceAbstract;
             b:name ?deviceNameLiteral.
}
```

図 28　部分クエリ B

- FILTER に「防水性能を持つ」と「機種名がリテラルである」という二つの条件を指定してクエリを組立て, 完成したクエリとを図 29 に示し, そのグラフを図 30 に示し, 検索結果を表 6 に示す

```
prefix a: <http://dbpedia.org/ontology/>
prefix b: <http://dbpedia.org/property/>
select distinct ?deviceNameLiteral from <http://dbpedia.org> ... where
{ ?deviceIRI a:abstract ?deviceAbstract;
             b:name ?deviceNameLiteral.
FILTER regex(?deviceAbstract,"waterproof","i")
FILTER isLiteral(?deviceNameLiteral)
}
```

図 29　完成したクエリ

図 30　完成したクエリのグラフ

表 6　検索結果（122 件）の一部

deviceNameLiteral
“Sony Xperia ZR”@en
“PRB 408”@en
“Xperia Z”@en
“Sony Xperia Z1”@en
“Sony Xperia Z1 Compact”@en
“Fujitsu Toshiba IS12T”@en
…

以上のような手順に従い，作成した情報要求 2 のクエリとクラフを図 31 と 32 に示し，情報要求 3 のクエリとクラフを図 33 と 34 に示す.

```
prefix a: <http://www.w3.org/2000/01/rdf-schema#>
prefix b: <http://purl.org/dc/terms/>
select distinct ?A
from <http://dbpedia.org>... where {
?X a:label ?A;
    b:subject <http://dbpedia.org/resource/Category:Manga_adapted_into_films>.
}
```

図 31　情報要求 2 の正解クエリ

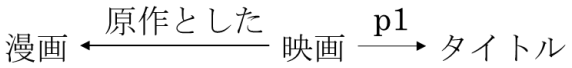

図 32　情報要求 2 の正解グラフ

```
prefix a: <http://www.w3.org/2000/01/rdf-schema#>
prefix b: <http://dbpedia.org/ontology/>
prefix c:<http://dbpedia.org/resource/>
prefix d:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix e: <http://dbpedia.org/dbtax>
select distinct ?A
from <http://dbpedia.org>...where
{ ?X a: label ?A;
      d: type e:Company;
      b: wikiPageWikiLink c:Computer_hardware;
      b: wikiPageWikiLink c:Smartphone.
}
```

図 33　情報要求 3 の正解クエリ

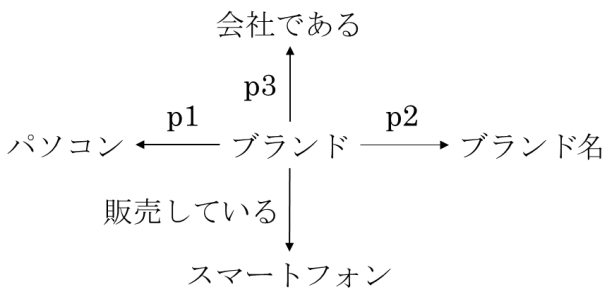

図 34　情報要求 3 の正解グラフ

4.3　実験結果

3 章で提案したグラフ生成処理を実装し, 4.2 で使用した情報要求例を用いて生成したクエリのグラフに沿って SPARQL クエリを組立てる手順例と結果の一部を以下に示す．情報要求から検索クエリのグラフへの変換ルールによっていくつかのグラフの候補が出力される可能性があるが，今回はトリプルが多い方を選んだ.

- 情報要求例「防水性能を持つデバイスの機種を知りたい」を入力し，生成したクエリのグラフを図 35 に示し,「[]」で囲まれているのは一つのグラフであり，その中の「()」で囲まれているのは一つのトリプルである．図 35 での「デバイス」の右にある「2」は，同じ名前の主語がある可能性があるため，それらを区別するために，プログラムで定義した 2 番目の文節から取り出して主語とする意味を表している．「p23」は，2 番目の文節から取り出した主語と 3 番目の文節から取り出した目的語を繋ぐ述語を意味している.

[(デバイス 2, 持つ, 性能), (デバイス 2, p23, 機種)]

図 35　情報要求例のクエリのグラフ

- クエリのグラフに基づいて,「デバイス 2」を「?X」とし，「性能」を「?A」とし，両者を繋ぐ述語「持つ」を「?p1」とする．「?A」の値に「waterproof」が含むと考えられるため，それを FILTER に指定して部分クエリ a で検索を行い，述語の候補 p1 とその出現頻度を提示する．部分クエリ a は図 36 のようになり，その検索結果を表 7 に示す.

```
select distinct ?p1, (count(?p1) as ?cp) from <http://dbpedia.org> ... where
{ ?X ?p1 ?A.
FILTER regex(?A,"waterproof","i")
}
group by ?p1
order by DESC(?cp)
```

図 36　部分クエリ a

表 7　部分クエリ a の検索結果

p1	cp
http://dbpedia.org/ontology/wikiPageWikiLink	459
http://dbpedia.org/ontology/abstract	444
http://dbpedia.org/ontology/wikiPageWikiLinkText	87
http://www.w3.org/2000/01/rdf-schema#label	36
http://dbpedia.org/property/shortsummary	18
http://dbpedia.org/ontology/wikiPageExternalLink	12
...	...

- 表 7 より説明文の意味を表している「http://dbpedia.org/ontology/abstract」を選び，部分クエリ a を完成する．「?X」と繋ぐ「機種」を「?Y」とし，「?Y」が文字列であるという条件を付けて，部分クエリ a と繋いで検索を行い，述語の候補 p23 とその出現頻度を提示する．部分クエリ b は図 37 のようになり，その検索結果を表 8 に示す．

```
prefix a: <http://dbpedia.org/ontology/>
select distinct ?p23, (count(?p23) as ?cp) from <http://dbpedia.org> ... where
{ ?X a:abstract ?A;
      ?p2 ?Y.
FILTER regex(?A,"waterproof","i")
FILTER isLiteral(?Y)
}
group by ?p23
order by DESC(?cp)
```

図 37　部分クエリ b

表 8　部分クエリ b の検索結果

p2	cp
http://dbpedia.org/ontology/wikiPageWikiLinkText	1557
http://www.w3.org/2000/01/rdf-schema#label	1274
http://dbpedia.org/ontology/abstract	1252
http://dbpedia.org/ontology/wikiPageOutDegree	444
http://dbpedia.org/ontology/wikiPageLength	444
http://dbpedia.org/property/name	127
...	...

- 表 8 より名前の意味を表している「http://dbpedia.org/property/name」を選び，部分クエリ b を完成すれば，クエリも完成する．完成したクエリを図 38 に示し，最後の検

索結果の一部を表 9 に示す.

```
prefix a: <http://dbpedia.org/ontology/>
prefix b: <http://dbpedia.org/property/>
select distinct ?Y
from <http://dbpedia.org> ...where
{ ?X a:abstract ?A;
      b:name ?Y.
FILTER regex(?A,"waterproof","i")
FILTER isLiteral(?Y)
}
```

図 38　完成したクエリ

表 9　結果（122 件）の一部

Y
“Sony Xperia ZR”@en
“PRB 408”@en
“Xperia Z”@en
“Sony Xperia Z1”@en
“Sony Xperia Z1 Compact”@en
“Fujitsu Toshiba IS12T”@en
…

前述の手順例に従い，残り 2 件の情報要求がそれぞれ提案手法で生成したグラフと作成したクエリ，そして正解として人手で作成したクエリとグラフを図 39 から 42 に示し，情報要求 1 を含めてそれぞれの比較結果を示す.

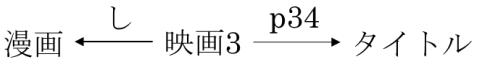

図 39　提案手法で生成した情報要求 2 のグラフ

```
prefix a: <http://www.w3.org/2000/01/rdf-schema#>
prefix b: <http://purl.org/dc/terms/>
select distinct ?A
from <http://dbpedia.org>... where {
?X a:label ?A;
    b:subject ?B
FILTER regex(str(?B),"manga","i")
FILTER regex(str(?B),"film","i")
}
```

図 40　提案手法で生成した情報要求 2 のグラフに基づいて作成したクエリ

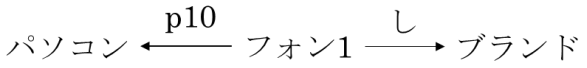

図 41　提案手法で生成した情報要求 3 のグラフ

```
prefix a: <http://dbpedia.org/ontology/>
prefix b: <http://www.w3.org/2000/01/rdf-schema#>
select distinct ?B
from <http://dbpedia.org> ... where
{ ?X a:abstract ?A;
      b:label ?B.
FILTER regex(?A,"computer","i")
FILTER regex(?A,"smartphone","i")
}
```

図 42　提案手法で生成した情報要求 3 のグラフに基づいて作成したクエリ

情報要求 1 の提案手法で生成したグラフと人手で作成したクエリのグラフが完全に一致したと判明した．また，提案手法で生成したグラフに沿って完成したクエリも人手で作成したクエリと同じだと分かった．

情報要求 2 の提案手法で生成したグラフと人手で作成したクエリのグラフにより，グラフ構造が同じパターンだと判明した．同じ主語と繋がる述語はマッチしたが，目的語がマッチしなかった．

情報要求 3 のグラフと結果について分析してみると，提案手法で生成したグラフは人手で作成したクエリのグラフの構造と一致したトリプルが 2 つあると分かった．両者の違いについては，まず人手で作成したクエリのグラフでは，「スマートフォン」を主語としたではなく，「ブランド」を主語とし，その「ブランド」が「スマートフォン」と「パソコン」それぞれに繋いでいる．そして，「ブランド」が会社であることも指定した．提案手法で生成した情報要求 3 のグラフに基づいて作成したクエリと人手で作成したクエリにより，図 42 では，「フォン 1」を「?X」とし，「パソコン」を「?A」とし，「ブランド」を「?B」とし，FILTER に「?A」に「computer」と「smartphone」が含むことを指定した．図 33 では，「ブランド」

を「?X」とし，「ブランド名」を「?A」とした．

提案手法に沿って得た結果の中に正解の結果と一致した比率（再現率）と，正解の結果と一致した数が実験の結果数に占めた比率（適合率）を表 10 に示す．再現率＝正解の結果と一致した数／正解の結果数，適合率＝正解の結果と一致した数／実験の結果数．表 10 によると，3 件の情報要求の中に再現率が 50%以上超えたのは 2 つであり，適合率が 50%以上超えたのは 1 つである．

情報要求 1 の提案手法に沿って組み立てたクエリは正解となる人手で作成したクエリと全てのトリプルが一致し，クエリも一致したため，再現率と適合率が 100%になった．

情報要求 2 の提案手法に沿って組み立てたクエリは正解となる人手で作成したクエリと一致したのは 1 つのトリプルであったため，再現率が相当に高くなった．また，実験の結果数が正解の結果と一致した数の 2 倍くらいあるため，適合率が少し低くなった．

情報要求 3 の提案手法に沿って組み立てたクエリは正解となる人手で作成したクエリと一致したのは 2 つのトリプルしかなかったが，正解の結果数が実験の結果数より少ないため，再現率が高くなった．一方，実験の結果数が正解の結果と一致した数の 19 倍くらいあるため，適合率が低くなった．

表 10　再現率と適合率

情報要求の番号	実験の結果数	正解の結果数	正解の結果と一致した数	再現率（%）	適合率（%）
1	122	122	122	100.00	100.00
2	2270	1382	1355	98.05	59.69
3	2530	161	128	79.50	5.06

5. 考察

前節で示した結果により，グラフの違いの再現率と適合率に対する影響という観点で考察を行う．

情報要求 1 の再現率と適合率がそれぞれ 100%になったのは，提案手法で生成したグラフと正解として人手で作成したクエリのグラフが完全に一致したので，二つのクエリが一致したためである．また，「名詞＋を＋動詞＋名詞＋の＋名詞＋を知りたい」のようなパターンの情報要求に対して有効だということを確かめるため，同じパターンであるが内容が違う情報要求で再び実験をした．「蛋白質を含む食品の名称を知りたい」を情報要求 4 として実験を行い，作成した正解グラフとクエリを図 43 と 44 に示し，提案手法で生成したグラフとクエリを図 45 と 46 に示す．情報要求 4 の提案手法で生成したグラフと正解として人手で作成したクエリのグラフが完全に一致したため，再現率と適合率もそれぞれ 100%になった．結論としては，「名詞＋を＋動詞＋名詞＋の＋名詞＋を知りたい」のようなパターンの情報要求に対して有効だと確認した．

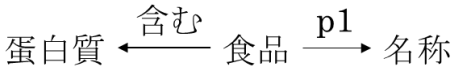

図 43　情報要求 4 の正解グラフ

```
prefix a: <http://dbpedia.org/property/>
select distinct ?deviceNameLiteral from <http://dbpedia.org> ... where
{ ?foodIRI a:protein ?ProteinQuantity;
          a:name ?foodNameLiteral.
FILTER isLiteral(?foodNameLiteral)
}
```

図 44　情報要求 4 の正解クエリ

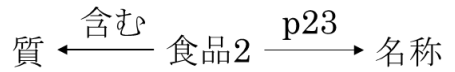

図 45　提案手法で生成した情報要求 4 のグラフ

```
prefix a: <http://dbpedia.org/property/>
select distinct ?B from <http://dbpedia.org> ... where
{ ?A a:protein ?Y;
     a:name ?B.
FILTER isLiteral(?B)
}
```

図 46　提案手法で生成した情報要求 4 のグラフに基づいて作成したクエリ

情報要求 2 では，生成したグラフが正解グラフと同じパターンになり，同じ主語と繋が

る述語はマッチしたが，目的語がマッチしなかったため，再現率が 100%に相当近くなったが，適合率がそこまで高くならなかった．

情報要求 3 の結果により，再現率が 3 件の中に二番目高いのは，提案手法で生成したグラフは人手で作成したクエリのグラフの構造と半分一致したためである．一方，適合率が低くなったのは，提案手法で生成したグラフが人手で作成したクエリのグラフより 2 つのトリプルが足りなかった上,「ブランド」と「スマートフォン」を繋ぐ方向も逆になったためである．それを解決するため，本手法の一部について再検討する必要がある．

以上の分析によって，本研究で提案した Linked Data クエリ構築支援のための日本語文に基づくグラフ構造の生成手法は以下の課題が残されている．

- トリプル生成ルール 3 では，動詞を述語にすると定めているが，対象データの出現頻度順で並んだ 60139 件の述語の中の 1000 件を調査した結果，末尾が動詞または動詞として使われている述語が 30 件しかないと発見したため，トリプル生成ルールを追加する必要があると考えられる
- 情報要求 3 の結果についての分析により，正解の結果と一致した比率が 79.50%になったが,「ブランド」と「スマートフォン」の繋ぎ方向が逆になった問題を解決するため，トリプル生成ルールの追加や修正をする必要があると考えられる

6. おわりに

本論文では， Linked Data の構造を指定する検索クエリの書き方を知っている開発者向けに，日本語文からスキーマ定義が不明な Linked Data のデータセットを検索するためのクエリのグラフ構造を生成する手法を提案し，その評価を行った．クエリの構築を支援するため，本研究では日本語係り受け解析器 Cabocha を使用し，その出力結果に基づいて SPARQL クエリのグラフ構造を自動的に生成することを実現した．

グラフ構造の自動生成を実現するため，この提案手法では日本語の情報要求の係り受け関係に基づいて文節結合ルールとトリプル生成ルールを定め，それをプログラムに変換した．更に，提案したグラフ構造の自動生成手法に沿って，生成したグラフを用いて組立てた SPARQL クエリを実行すると，得られた結果の中にどれだけがグラフ生成ルールを使わずに人手で検索を行って得られた結果と一致したかについて確認した．結果としては，本研究で提案したグラフ生成ルールが「名詞＋を＋動詞＋名詞＋の＋名詞＋を知りたい」のようなパターンの情報要求に対して有効性が高いと分かった．一方，トリプル生成ルールの修正と追加についての検討が課題として残された

謝辞

研究生の期間を含めて二年半の間に，本研究を進めるにあたり，研究の進め方から論文の書き方まで，研究に関する様々なことをご指導頂いた阪口哲男先生に心より感謝致します．また，合同ゼミの場で様々なコメントを頂いた森嶋厚行先生，並びに杉本重雄先生，永森光晴先生に感謝致します．そして，合同ゼミで発表した際に協力して頂いた森嶋研究室，杉本・永森研究室の皆様に感謝致します．

参考文献

[1] 加藤文彦, 川島秀一, 岡別府陽子, 山本泰智, 片山俊明. オープンデータ時代の標準 Web API SPARQL. 株式会社インプレス R&D, 2015, p. 14.
[2] 同上, p. 21.
[3] W3C, “SPARQL 1.1 Query Language”. https://www.w3.org/TR/sparql11-query/, (参照 2016-12-13).
[4] “where does my money go?”. http://app.wheredoesmymoneygo.org/, (参照 2016-12-13).
[5] “さばえぶらり”. http://atr-c.jp/burari/product/oldmap/sabae.html, (参照 2016-12-13)
[6] “LOD チャレンジ Japan 2013”. http://lod.sfc.keio.ac.jp/challenge2013/, (参照 2016-12-20).
[7] 西出頼継, 本間維, 永森光晴. 日本の Open Data 活用を目的としたデータセットのスキーマ分析とリンク関係の調査. 情報処理学会第 112 回情報基礎とアクセス技術研究会 (IFAT), vol.2013-IFAT-112, NO.4.
[8] “○○危険地帯”. https://github.com/uedayou/dangerzone-sparql, (参照 2016-12-22).
[9] “大阪市の施設情報”. http://linkdata.org/work/rdf1s933i?key=#work_information, (参照 2016-12-27).
[10] Jens Lehmann, Robert Isele, et al. “DBpedia-A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia.” Semantic Web, 2015, vol. 6, no. 2, p. 167-195.
[11] 安永ゆい, 森嶋厚行. 構造/テキストWebデータを対象としたPay-as-you-goスタイルの問合せ構築支援. 筑波大学図書館情報メディア研究科修士論文, 2015.
[12] Tsunagu Honma, et al. "Extracting description set profiles from RDF datasets using metadata instances and SPARQL queries." Proceedings of the 2014 International Conference on Dublin Core and Metadata Applications. Dublin Core Metadata Initiative, 2014, p. 109-118.
[13] Nilsson M. “Description set profiles: A constraint language for dublin core application profiles.” Latest version: http://dublincore. org/documents/dc-dsp, 2008.
[14] Atsuko Yamaguchi, et al. "An intelligent SPARQL query builder for exploration of various life-science databases." Proceedings of the 3rd International Conference on Intelligent Exploration of Semantic Data, 2014, Vol. 1279, p. 83-94.
[15] Fei Li, Jagadish H V. "Constructing an interactive natural language interface for relational databases." Proceedings of the VLDB Endowment, 2014, vol. 8, no. 1, p. 73-84.
[16] Taku Kudo. "CaboCha: Yet Another Japanese Dependency Structure Analyzer." Technical report, Nara Institute of Science and Technology, 2004.