社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－ＦＩ－78 (11)
2005－ＤＤ－49 (11)
2005／3／25

Prologを使ったRDFデータからのユーザ指定による文書構築

花川賢治† 天笠俊之‡ 波多野賢治‡ 宮崎純‡ 植村俊亮‡

† 大阪府立工業高等専門学校 電子情報工学科 ‡ 奈良先端科学技術大学院大学 情報科学科

本発表では、事実の集合を格納するデータベースからユーザの要求に応じた構造化文書を生成する処理について論じる。代表的な事実についての表現形式として、データベースと構造化文書が存在する。両者は対照的な性質を持ち、前者が意味的に独立した要素の集合であり、唯一の存在であるのに対し、後者は要素間の関係が木構造化され、その構造は利用者の参照の条件に応じて多数の種類が存在し得る。多数の種類の構造化文書を効率的に処理するためには、宣言的に構造化文書の仕様を記述するための言語と、データベースから構造化文書への変換を行う汎用の処理系が重要になる。本発表では、構造化文書の仕様記述言語に一階述語論理式を用い、Prologを拡張した処理系を実装する。

User Specified Document Construction from RDF Data Using Prolog

Kenji Hanakawa† Toshiyuki Amagasa‡ Kenji Hatano‡ Jun Miyazaki‡ Shunsuke Uemura‡

†Department of Electrical Engineering and Computer Science, Osaka Prefectural College of Technology
‡Graduate School of Information Science, Nara Institute of Science and Technology

We discuss a process of producing structured documents from a databases which contains facts. The usual methods of representing facts are a database and a structured document. They are very different. A database consists of independent elements and uniquely exists. In contrast, structured documents have tree structures where connections between elements have meanings and types of them are very diverse. In order to treat the diverse structured documents effectively, we propose a deductive language for specifying a tree structure and a general language processor for translating from a database to a structured document. In this language, the tree structure is specified in a first order predicate logic formula, and the language processor is implemented by extending a Prolog interpreter.

1 はじめに

本発表では、事実の集合を格納するデータベースからユーザの要求に応じた構造化文書を生成する処理について論じる。ここで対象とする事実の集合のデータベースは、近年注目を集めているRDFで書かれたインターネット上の情報リソースのような、単純な日常的な情報を集めた比較的小規模なものを想定している。

RDF[4]は、主語、述語、目的語からなる三つ組みの文による情報の表現形式である。RDFはWWW上のリソースを記述するためのものと定義されているが、WWW上のリソースについての情報とそれ以外の実体についての情報の間に本質的な差異はないので、RDFの記述対象をWWW上のリソースに限定することは無意味である。むしろ、RDFの本質は表現形式の単純さにあり、そのために幅広い人々が身近な事実情報を容易に記述できるようになったことが重要である。RDF、もしくはRDFのような簡単な形式のテキストによるデータベースは、身近な情報を日常的にデータベース化する行為を普及させ、その結果、カジュアルユーザによって生成されたデータベースはインターネットリソースとして豊富に入手可能になると予想される。

RDFは、木構造を持つ構造化文書としての側面も持っている[1]。RDFの文脈では、構造化文書はタグを含むXML文書のことを指すが、本研究では、ある程度深い木構造を持つ文書が構造化文書である

と考える。ここでの構造化文書には、XML文書も含まれるが、それ以外の形式、たとえばLaTeX、roff、HTML、平坦テキストも含まれる。構造化文書の木構造の各ノードは文書要素に対応し、ノードに付くラベルは述語であり、それは実際の文書では見出しとして表現されると考える。したがって、本発表で構造化文書の木構造を図式表現するときには、ノードのラベルにはタグの種類ではなく、実際の文書に含まれる見出しか述語を書く。

一般に木構造による情報の構造化は情報の参照を目的としたものであり、利用者の参照時の状況に依存して最適な構造は異なるはずである。しかしながら、RDFを構造化文書化したRDF/XMLの形式は多様な木構造を許していない。

このような問題は、RDF/XMLだけでなく、広く存在する問題である。現在の計算機システムにおいて、構造化文書、ファイルシステム、階層型メニューなど、情報の木構造化は広く応用されている。しかし、一般に、木構造の生成が情報を格納する時点で行われるため、ユーザが参照する時点において必ずしも最適な構造とはなり得ず、その能力を十分に活かしきっていない。

この問題を解決するためには、多様な構造化文書の仕様を効率的に記述する言語とデータベースと構造化文書の間を変換する処理系の実現が必要である。本発表では、構造化文書の記述言語に一階述語論理式を用い、処理系をPrologインタプリタを拡張して実装する。ただし、本発表では、データベースから構造化文書への変換のみを扱い、構造化文書からデータベースへの変換については述べない。

一階述語論理式による構造化文書の記述は、宣言的で非常に簡潔である。また、利用者が望む多様な文書構造を完全に記述する能力を持つことが期待できる。したがって、利用者は非常に小さな労力で、事実のデータベースから自由に要求した構造を持つ構造化文書を得ることができる。

ここで提案する言語が、全くの新規の言語ではなく既存のPrologのわずかな拡張であることも、重要な長所である。近年、多数の構造化文書に関する言語が提案されているが、利用者にとって、そのような言語の不安定な仕様を学習することが多大な負担になっていると思われる。Prologのような、設計思想、言語仕様、実装が十分に評価済みで信頼性が高い言語を採用したことは、利用者にとっても大きなメリットがあると考えられる。

2 多様な構造化文書の処理

代表的な事実の表現形式として、データベースと構造化文書が存在する。まず、対照的な性質を持つ両者の特徴について述べる。データベースは、事実を表現した独立性の高い要素の集合である。すなわち、個々の要素の持つ意味は他の要素の影響を受けない。それに対し、構造化文書は、要素間の相互関係に基づく木構造を持つ。木構造内の要素間の接続には意味がある。データベースは実世界の素直な反映で、唯一の安定した存在であることが望まれる。一方、構造化文書は参照時に個別の利用者の状況に適した多数の種類が得られることが望まれる。XML関係の研究では、データベースと構造化文書を明確に区別しない立場を取っているように思えるが、本研究では、両者の違いが重要であると考える。

以下に、例を使って、複数種類の構造化文書が構成可能であることを示す。図1に示すような6個の図形があったときに、それを (主語, 述語, 目的語) の三つ組みの形式で表わすと図2のデータベースのようになる。それぞれの三つ組みは事実を表わし、三つ組みの並び順には意味はない。それから構成可能な構造化文書の例を図2に示す。ここでは2種類を示すにとどめたが、さらに他の構造も構成可能である。両者は6個の図形の色と形の情報に基づき構成されているが、木構造が異なる。どちらの木構造が望ましいかは、一概には言うことはできなく、利用者が持つ背景や参照時の状況に依存する。

唯一のデータベースと多種類の構造化文書に関係した基本的な処理として以下が考えられる。

1. 問い合わせ
   一つまたは複数の事実についての情報を引き出す。
2. 構造化文書の参照
   関連のある複数の事実の要素を木構造に配置して利用者に示す。
3. 異なる種類の構造化文書間の変換
   ある構造化文書から、それと同じ内容を持つ構造の異なる構造化文書を作る。

現在提案されている多くのXML処理系では、構造化文書を直接的に処理し、処理を手続き的に記述する方法が採用されている。しかしながら、この方

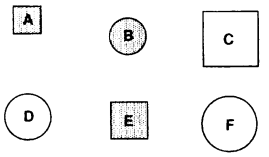

図 1: 実世界の例

法では、多種類の構造化文書を効率的に扱うことができない。

問い合わせにおいては、XQuery のような XML 文書を走査して問い合わせ文との照合を行う言語では、文書の木構造の種類に応じて問い合わせを記述する必要がある。また、異なる構造の構造化文書への変換においては、XSLT[2] のような方法では、変換元と変換先の組み合わせごとに処理を記述するため、構造化文書の種類が多いと、必要な変換処理は莫大になる。

XML 処理系とは対照的に、本研究では、構造化文書の仕様を宣言的に記述することと、データベースと構造化文書を明確に区別し、それらの間で変換を行うことで、上に述べた基本的な処理を実現する (図 3)。

この方法では、問い合わせはデータベースに対して行う。したがって、問い合わせ文は構造化文書と無関係に作成することが可能である。構造化文書の参照は、データベースから利用者が必要とする事実の要素を選択し、利用者が与えた仕様に従い構造化文書を動的に生成することで実現する。異なる種類の構造化文書間の変換は、変換元の構造化文書から事実の要素の集合 (データベースの部分集合) を得て、それから与えられた仕様に従い変換先の構造化文書を再構成する。

この方法では、構造化文書の種類の増加により記述が必要になるのは、それぞれの構造化文書に対応した仕様記述だけである。

3 構造化文書の宣言的な仕様記述

本研究では、以下のような文書の木構造の意味論を採用する [3]: **文書の木構造のノードは述語を意味し、根からあるノードへのパスはその上にあるノードの論理積が真であることを意味する。**

したがって根から葉に至るパスは、質問全体が真

データベース

```
(a, type, object), (a, shape, square), (a, color, black),
(b, type, object), (b, shape, circle), (b, color, black),
(c, type, object), (c, shape, square), (c, color, white),
(d, type, object), (d, shape, circle), (d, color, white),
(e, type, object), (e, shape, square), (e, color, black),
(f, type, object), (f, shape, circle), (f, color, shite),
(square, type, shape), (circle, type, shape),
(black, type, color), (white, type, color)
```

構造化文書の例 1

```
shape square
        color black
                a
                e
        color white
                c
shape circle
        color black
                b
        color white
                d
                f
```

構造化文書の例 2

```
color black
        a square
        b circle
        e square
color white
        c square
        d circle
        f circle
```

図 2: データベースと構造化文書の例

であることを意味する。

この意味論では、述語とノードが対応し、ノードのラベルは述語を表現したものである。実際の文書では、ノードのラベルは見出しであり、多くの場合名詞句である。一般に述語を自然言語で表わすと文になるので、自然言語の構文カテゴリという観点では一致しない。しかし、実際の文書の見出しは簡潔さのために省略した表現が行われており、本来文で表わされるべき表現が名詞句で表わされていると考えることができる。

Prolog のような一部のシステムを除き、多くのデータベースシステムは直接的には一階述語の表現形式を使っていない。しかし、事実に関する単純なデータベースの記述形式は一階述語に変換することが可能である。図 2 に示した、RDF の三つ組みによる事実の記述は、図 4 のような事実の集合に書き換えることができる。

この意味論に従えば、以下の手順により一階述語論理式とデータベースから構造化文書を構成することができる。

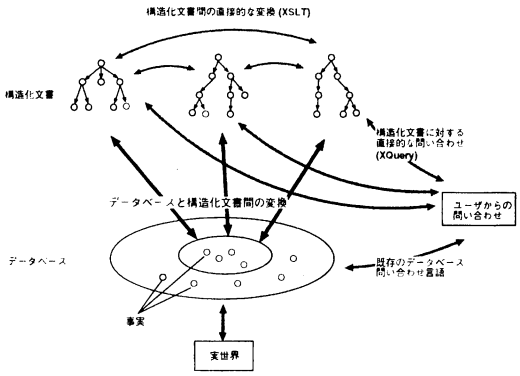

図 3: データベースと構造化文書の処理

```
object(a).      shape(a,square).     color(a,black).
object(b).      shape(b,circle).     color(b,black).
object(c).      shape(c,square).     color(c,white).
object(d).      shape(d,circle).     color(d,white).
object(e).      shape(e,square).     color(e,black).
object(f).      shape(f,circle).     color(f,white).
shape(square).  shape(circle).
color(black).    color(white).
```

図 4: データベースの述語表現の例

1. 選言標準系への変換
   与えられた一階述語論理式をリテラルの論理積の論理和の形式に変換し、リテラルの論理積に含まれるすべての変数に定数が代入された解を求める。それぞれの解から、それに含まれる述語をノードに置き換えリンクで繋ぐことによりパスを作る。

2. 兄弟ノードの結合
   すべてのパスの端を根とすることで、木を生成する。兄弟に同一の述語が含まれるばあいには、それらを一つに結合する。同一の述語とは、述語記号、引数の個数、すべての引数の値が一致する述語のことである。しかしながら、述語の完全な一致は、任意の木を構成するのに条件が厳しすぎるので、指定した引数を一致の条件から除くことを許す。図5の上の木は論理式 $shape(X,S) \wedge color(X,white)$ から構成される木である。shape の第1引数を一致の条件から除くように指定すると、図5の下の木のように $shape(d,circle)$ と $shape(f,circle)$ を結合することができる。

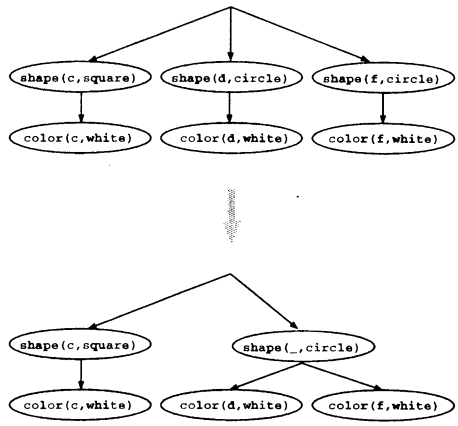

図 5:

3. 書式付き出力
   述語をノードとする木構造から目的とする構造化文書を生成するために、深さ優先探索による走査を行い、ノードを訪問したときに、それに対応した出力を行う。そのときに、指定された書式に従い述語から出力する文字列への変換を行う。多くの構造化文書が要素の開始と終了に表示が置かれることに対応して、始点書式と終点書式を定義する。始点書式に基づく出力はノードを訪問したタイミングで、終点書式に基づく出力はノードから去るタイミングで行われる。

この一階述語論理に基づく構造化文書の仕様記述には、二つの特徴がある。一つは、任意の論理式からそれに対応した木構造を生成することができる点である。逆に、利用者にとって意味のある任意の文書構造を一階述語論理式で記述することも可能であると思われる。もう一つの特徴は、述語という抽象化した要素を利用するので、データベースからの独立性が高い点である。

4 Prolog による構造化文書の自動生成

ここでは、Prolog を拡張した処理系の実装について述べる。

この処理系に要求される基本機能として以下が考えられる。

1. 任意の論理式による入力

2. リテラルの積による出力

Prologでは、これらの機能を容易に実現できる。論理式による入力は、Prologの質問に相当する。本来のホーン節の質問はリテラルの論理積であるが、多くのProlog処理系では、OR(;)を質問に含めることができるので、複雑な論理構造を持つ質問を記述することが可能である。リテラルの論理積の形式で解を得るために、図6に示すような、call_を定義する。Prologには、質問を処理するための組み込み述語callがあり、単に質問を引数に与えcallを呼び出すと、インタプリタのトップレベルでその質問をしたのと同じ結果が得られる。call_はPrologの組み込み述語callと同様に第1引数に与えられた質問を処理する。それに加えて、成功した述語のリストを第2引数と同一化する。

call_は再試行するたびに異なったパスを第二引数に返す。多くのProlog処理系にある組み込み述語findallを用いることによりcall_が失敗するまで再試行させ全解を求め結果をリストに格納することができる。パス自体がリストであるので全解リストは二重のリストになる。その二重のリストから兄弟ノードの結合処理を根から再帰的に実行することにより、文書木を生成する。

質問の述語には、兄弟ノードの結合のときに不問にする引数、始点書式、終点書式、の3種類の情報を付加する必要がある。そこで、以下のようにPrologリストを:で区切って、本来の述語の後ろに置く構文を定義する。

述語:
兄弟ノードの結合のときに不問にする引数リスト:始点書式リスト:終点書式リスト

兄弟ノードの結合のときに不問にする引数リストは整数を要素とするリストである。リストに含まれる番号の引数は、兄弟間の結合時に値が不一致であっても、一致したかのように処理される。

書式リストは、アトム、数、変数を要素とするリストである。リストの要素は連結して出力される。始点書式リストはノードを訪問したときに、終点書式リストはノードを去るときに出力される。

これらのリストは省略可能である。すべてを省略、つまり述語のみのばあいには、兄弟間の結合時には完全な述語の一致が調べられ、訪問時に述語がwriteにより、そのままの形式で出力される。

```
call_((X;Y), Path):- !,
        (call_(X, Path); call_(Y, Path)).
call_((X,Y), Path):- !,
        (call_(X,Path1), call_(Y,Path2)),
        append(Path1, Path2, Path).
call_(X, [X]):-
        call(X).
```

図6: 規則call_の定義

```
call_(Name, Path):-
        ::(Name, Definition),!,
        call_(Difinition, Path).
```

図7: マクロ機能の追加

質問の部分を再利用するために以下の構文を持つマクロ定義が扱えるように拡張を行う(図7)。

マクロ名(引数, ...) :: マクロの本体

マクロは述語と同様に引数を与えて呼び出すことができる。マクロが呼ばれると、:: の右辺のマクロ本体が質問される。マクロ本体は述語と論理演算子から構成される。マクロ本体にマクロ呼び出しを含めること、つまりマクロの入れ子も可能である。

マクロ定義は一般のPrologの規則の定義と解を求める機能については同じであるが、*call_*が返すパスに関しては違いが現れる。規則のばあいは、右辺の構造はパスに反映されないため、階層的な構造を得ることができない。両者の違いを説明した例を図8に示す。*rule*を質問したばあいは、長さ1のパスが得られ、出力される木も高さが1となり、平坦な構造しか得られない。*macro*を質問したばあいは、右辺の*ward*と*ward_town*に対応した長さ2のパスが得られ、高さ2の木構造が得られる。

図2に示した構造化文書は、以下のPrologの質問により生成することができる。

```
?- shape(X,S):[1]:["shape ",S,"\n"]:[],
   color(X,C):[1]:["\t color ",C,"\n"]:[],
   true:[]:["\t\t",X,"\n"]:[].

?- color(X,C):[1]:["color",C,"\n"],
   shape(X,S):[]:["\t",X," ",S,"\n"]:[].
```

5 ケーススタディ

実際に処理系を実装し、構造化文書の生成を試み、多様な文書構造を簡潔に指定することができること

```
(A) 規則の定義と呼び出した結果

rule(X,C) :- color(C), color(X,C).

?- rule(X,C).

rule(a,black)
rule(b,black)
rule(e,black)
rule(c,white)
rule(d,white)
rule(f,white)


(B) マクロの定義と呼び出した結果

macro(X,C) :: color(C), color(X,C).

?- macro(X,C).

color(black)
        color(a,black)
        color(b,black)
        color(e,black)
color(white)
        color(c,white)
        color(d,white)
        color(f,white)
```

図 8: 規則とマクロが返すパスの違い

を確認した。ここでは、その一例を報告する。

実行環境には、Linux と SWI-Prolog を使った。SWI-Prolog には、Prolog-script という機能があり、質問を書いたテキストファイルを直接実行することができる。そのことを利用して、マクロと質問を Prolog インタプリタとの対話処理ではなく、テキストファイルに記述するようにした。

情報源は、WWW 上のオンラインのレストランガイド、*http://gourmet.yahoo.co.jp/* である。このサイトにはレストランごとの HTML ファイルが置かれている。HTML ファイルには、カテゴリ、所在地、住所、電話番号、最寄り駅、営業時間、主なメニューなどの項目が含まれている。それぞれに *category*、*location*、*address*、*tel*、*station*、*open*、*menu* などの述語を定義した。すべての述語は 2 個の引数を持ち、第 1 引数はレストランの名前、第 2 引数は属性値である。そのほかに区と街を関係付ける *ward_town* も定義した。これらは、RDF の三つ組みの形式でも記述することが可能であり。もとのデータの形式は HTML であったが、将来、同様の情報が RDF データとして存在することが予想される。

レストランのガイドの仕様は以下のようである。

1. 属性による分類を指定する。
   ある区を指定して、その区内の街ごとにレストランを表にまとめる。

```
<table border=1>
<tr><td>まちの名前<td><table border=1>
  <tr><td>レストランの名前<br>
    そのレストランの住所<br>
    メニュー
      <ol>
      <li>メニューの項目</li>
      </ol>
    <br>
  </td></tr>
</table></td></tr>
</table>
```

図 9: 出力する HTML ファイルの形式

```
sort_by_town(T,R)::
  true:[]
      :['<tr><td>',T,'</td><td><table border=1>']
      :['</table></td></tr>'],
  location(R,T):[]:['<tr><td>']:['</td></tr>'].

page(R)::
  true:[]:[R,'<br>']:[],
  (address(R,Adr):[]:[' 住所', Adr, '<br>']:[];
   true:[]:[' メニュー:<ul>']:['</ul>'],
   menu(R,Menu):[]:['<li>', Menu, '</li>']:[]).

?- true:[]:['<table border=1>']:['</table>'],
   ward_town(' 中央区',T):[]:[]:[],
   sort_by_town(T,R),
   (category(R,' イタリア料理'):[]:[]:[];
    category(R,' フランス料理'):[]:[]:[]),
   page(R).
```

図 10: レストランガイドブックを生成するための質問

2. 出力するフォーマットを指定する。
   レストラン名、住所、メニュー (箇条書き) を出力する。

3. 選択する条件を指定する。
   イタリア料理またはフランス料理のレストランだけを選ぶ。

また、出力する HTML ファイルは図 9 に示すような形式とする。

以上の要求を満たす構造化文書を作成するための質問を図 10 に示す。

まず、最初の要求は、*ward_town* と *sort_by_town* マクロにより実現する。*sort_by_town* マクロでは、街に対応した行を作成し、その中にレストランを格納する入れ子の表を格納している。

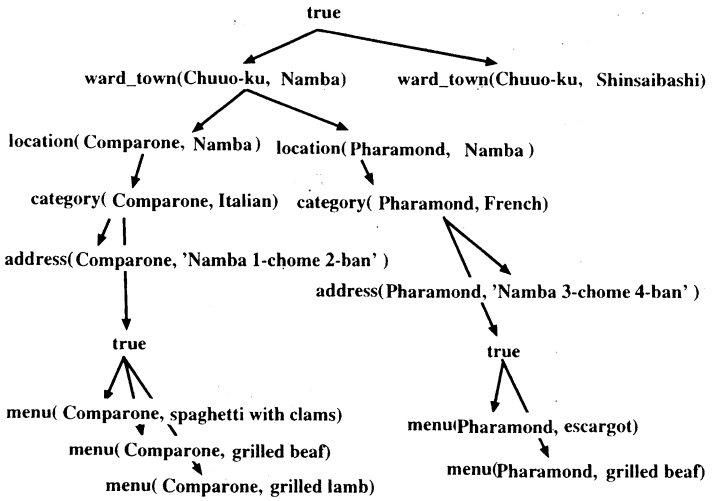

図 11: レストランガイドブックの文書木構造

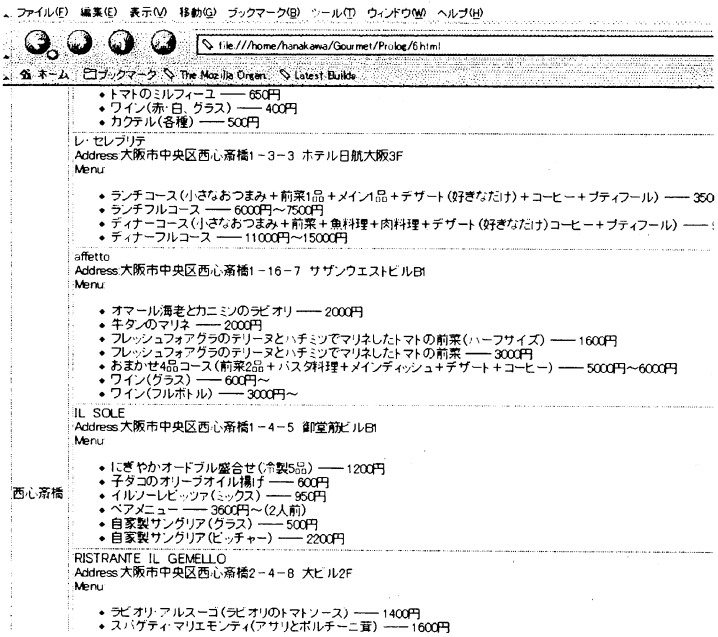

図 12: レストランガイドブックの WWW ブラウザ出力

次のフォーマットの要求は *page* マクロで実現する。*page* マクロの引数には具体的なレストランの名前がバインドされ呼び出される。*page* マクロの最初の true はレストランの名前を出力する目的のためにあり、次の *address* でそのレストランの住所を得て、br タグを加えて出力している。その次の true はメニューの項目を箇条書きにするための ul タグを出力するためのものである。最後の menu の行でメニュー項目に li タグを付加して出力する。一つのレストランに複数のメニュー項目があれば、一度に呼び出しに対し、*menu* はその数だけ再試行され、メニュー項目の出力が繰り返される。このケースでは、page マクロの最初の true から始まる行と address から始まる行の末尾のセミコロンはカンマに置き換えても動作は変わらない。

最後のレストランを探す条件は、質問の中間に置かれた `(category(R,'イタリア料理'):[]:[]:[];category(R,'フランス料理'):[]:[]:[]` により与えられている。これによりカテゴリがイタリア料理かフランス料理のレストランだけが出力される。この論理式は OR 演算子を含むが、すでに R にレストラン名がバインドされて呼び出されるため、枝分かれは発生しない。

結局、出力される HTML ファイルは、街の行が格納された一つの表である。街の行はレストランの行を含む。レストランの行は、レストランの名前、住所、メニューのリストを含む。

この質問から生成される文書木とブラウザでの表示を図 11 と図 12 に示す。

6 おわりに

代表的な事実についての表現形式としてのデータベースと構造化文書が異質な性質を持ち、構造化文書には多数の種類が必要であることを述べた。多数の種類の構造化文書を効率的に処理するために、宣言的に構造化文書の仕様を記述するための言語とデータベースから構造化文書への変換を行う汎用の処理系を提案した。提案された言語は一階述語論理に基づき、処理系は Prolog インタプリタを拡張することで実装した。ここで提案した言語は、宣言的な記述ができるので、非常に簡潔な表現が可能になる。そのことを、オンラインレストランガイドの構造化文書生成を行うことで確認した。実際の利用者が多数の種類の構造化文書を要求することと、それらを完全に本言語で記述できることを実験的に検証することが今後の課題である。

謝辞

本研究の一部は，科学研究費補助金 (基盤研究 (A)(2) 課題番号: 15200010) の支援による．ここに記して謝意を表す．

参考文献

[1] D. Beckett: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210 (2004)

[2] J. Clark: http://www.w3.org/TR/1999/REC-xslt-19991116 (1999)

[3] K. Hanakawa, T. Amagasa, K. Hatano, J. Miyazaki, S. Uemura: A Predicate Based Query Language for User-Specified Document Organization. SCI 2004. (2004)

[4] F. Manola, E. Miller: http://www.w3.org/TR/2004/REC-rdf-primer-20040210 (2004)