RDB を利用したタグ付きコーパス検索支援環境の構築

工藤 拓　　松本 裕治

奈良先端科学技術大学院大学 情報科学研究科

〒 630-0101 奈良県生駒市高山町 8916-5

{taku-ku,matsu}@is.aist-nara.ac.jp

言語データ (コーパス) 主導の統計的なモデルが発達する一方で, コーパスを柔軟に検索し, 個々の事例を閲覧する環境, システムはあまり進歩が見受けられない. 我々は, 現在の検索環境をより豊かにするために, 関係データベース (RDB) を用い, 構文情報等の高度なタグが付与されたコーパスを柔軟に検索する環境を構築した. さらに, RDB に格納されたデータは, 一般に SQL を用いて検索するが, SQL の知識が無くても, GUI 環境で簡単にクエリを作成するツールを作成した. 本稿では, 具体例を交じえながら, コーパスを RDB に格納する手法, RDB と SQL を用いた検索の説明, さらに作成したツールの紹介を行う.

キーワード: 関係データベース, コーパス, 検索, SQL

A Flexible Query Environment for Annotated Corpora using Relational Database

Taku Kudoh　　Yuji Matsumoto

Graduate School of Infomation Science, Nara Institute Science and Technology

8916-5 Takayama, Ikoma Nara 630-0101 Japan

{taku-ku,matsu}@is.aist-nara.ac.jp

Despite of recent advances in corpus-based approaches, little progress has been made for a flexible query environment for annotated corpora to view examples on demand. In order to enrich the conventional query environment, we develop a flexible and easy query environment for syntactically annotated corpora using Relational Database (RDB). In addition, we prepare a GUI tool which enables users to create SQL query easily and intuitively even if they have no knowledge of SQL. In this paper, we present a method to search syntactically annotated corpora using RDB and SQL. We also present some query examples of our GUI tool.

Keywords: Relational Database, Corpora, SQL, Query Environment

1 はじめに

1990 年代以降, 言語データ (コーパス) に基づくデータ主導型のアプローチの有効性が多くの研究者や技術者に認識されるようになった. また, 確率的言語モデルや, 機械学習に基づくアプローチにより, 広範囲の適用能力を持ち, 頑健で, しかも高性能な自然言語システムが構築できるようになった.

確率的言語モデルや機械学習といった「理論」は急速に発展して行く一方で, 実在するコーパスを「取り扱う」基礎的な技術は, 少なくとも言語処理の分野においてほとんど進歩していない. 例えば, コーパスの中の任意の部分木を検索するという基本的な処理一つを取ってみても, 汎用的かつ統一的な手法, システムは提案されていない.

検索ツールとして最も有名で頻繁に使用されるコマンドに grep が挙げられる. grep は 単語を key として検索するには十分であるが, 構文解析済みのデータから任意の部分木を検索したり, より高度な言語情報が付与されたコーパスを検索するには役不足である.

コーパスに対する高度な検索, つまり, コーパス中のある条件を満たすエントリの頻度を数えたり, 構文情報や係り受け関係を含む検索は, その度に使い捨てのプログラムを作成して対処しているのが現状である.

我々は, 現状の貧弱な検索環境を少しでも豊かなものにするために, RDB を利用し, 柔軟にかつ簡単にタグ付きコーパスを検索する環境を構築した. 本稿では, 具体例を交じえながら, 関係データベース (RDB) を用いた検索の説明と作成したツールの紹介を行う.

2 コーパスと検索技術

我々言語処理の研究者は, 以下のような検索要求にしばしば遭遇する.

(1) 「奈良」 が含まれる文を列挙せよ
(2) 接尾辞の前後に来る品詞や語彙を調べよ
(3) 任意の bigram 品詞連接頻度を計算せよ
(4) ある動詞が取りうる格フレームの種類を列挙せよ
(5) ある特定の助動詞を持つ動詞句にかかる副詞を列挙せよ
(6) 「を」と「に」を格に持ち, 「に」格は人名である動詞を列挙せよ

さて, このような要求が与えられた場合, どのように解決したら良いだろうか?

(1) は, grep といった単純な検索ツールにより解決できる. grep は, 線形探索であるため, 高速に検索した場合は, Suffix Array[3, 6] の使用が考えられる. また, 単純な文字列を key として, その文や前後の文脈を表示するツールとして KWIC (Key Word In Context) がある. (1) は, KWIC の一般的な検索要求である.

(2) も, 行指向のコーパス[1]であれば, grep を使い解決できる[2]. また, Suffix Array を用いた場合でも, 「接尾辞」を key に検索し, その前後のコンテキストを表示すれば良い. しかし, S 式といった 行指向以外のコーパスの場合, 単に周辺のコンテキストを表示するだけに終わり, 検索結果の可読性が悪くなる可能性がある.

(3) の場合, 一般的な UNIX のコマンドの組み合わせで解決する事は困難である. このような場合, Perl や Ruby といったスクリプト言語を用い, 頻度を数えるプログラムを作成する事で解決する場合が多い.

しかし, 検索要求がある度に使い捨てのスクリプトを生成する必要があるため効率が悪い. また, ある程度のプログラミング能力が必要となるため, 「簡単に」検索する環境とは言い難い.

(4),(5),(6) の場合, 一般的な UNIX のコマンドの組み合わせで解決する事は絶望的である. また, これらの検索要求を解決するには, 係り関係といった非線型な順序関係を考慮する必要があるため, 単純な文字列検索では解決しにくい. さらに, コーパスに付与された係り受け関係を考慮しながら検索する事はもちろん, 「を」と「に」の順序は規定しない所が検索要求をさらに複雑にしている. これらの処理も, スクリプト言語を用いる事で, 理論的には実現可能であるが, 検索要求が複雑であるため, プログラムが肥大化し, それ自身を作成するのに多くの手間や労力を必要とする. また, 検索要求は日々変化する一方で, その度に検索プログラムを書いていたのでは, 本来の研究の妨げになり, 研究そのものの生産性を悪くする.

このように, コーパスを検索するという言語処理において極めて基本的な処理は, 未だに低次な技術の組み合わせで実現されているのが現状である. 何らかの検索システムを作成したとしても, ある特定の検索に特化したシステムになりがちで[3], あらゆる要求に耐えうる汎用的なシステムの構築が急務となっている.

3 RDB を利用したコーパス検索

3.1 関係データベース (RDB)

データベースとは, 「秩序だった」データの集まりと定義できる. 「秩序だった」データを集めておく方式には種々の方式があるが, 今日最も広く使用されて

[1] 京都大学コーパスや RWCP コーパスなど
[2] GNU の grep であれば, grep -3 +3 とすれば 前後 3 行のコンテキストを表示する
[3] KWIC も前後の文脈のみ表示する処理に特化したツールであると言える

いるデータベースとして, 関係データベース (RDB) がある. 特に, 近年のインターネット上での電子商取引等で RDB の需要が高まり, 注目されるようになってきた. RDB にデータを蓄積したり, RDB からデータを検索抽出したりするソフトウェアを, RDBMS と呼び, 代表的なものに Oracle, MySQL, PostgreSQL などがある.

RDB は, 複数の「テーブル」の集合と捉える事ができる. 各「テーブル」は「列 (カラム)」と「行 (タプル)」から構成され, 各列にはフィールド名が付与される. 各行は現実世界の一つの実体 (エントリ) が格納される. 「テーブル」「列」「行」は, 和, 差, 直積, 選択, 射影という 5 つの関係演算により, 用途に応じて柔軟に変形すると事が可能である. これらの操作の具現化の一つとして SQL (Structured Query Language) があり, RDB 中のデータの検索や格納に使用されている.

3.2 コーパスの格納方法

RDB は, 複数のフラットなテーブルのみを提供するのに対し, 実在するコーパスは, 単純な単語の並びといった線形関係はもちろん, 構文情報や係り関係といった階層的な関係も記述されている. 両者のギャップをいかに解決するか, つまり, コーパス中の高度な情報ををいかにして RDB に格納するかが一つの大きな課題となる.

我々は, 文献 [1] を参考にしながら, コーパスを階層構造で記述された一つのデータとみなし, 個々の階層を一つのテーブルに対応させる事で, この問題を解決した. 例えば, 各テーブルとして「形態素」「文節」「文」「段落」等が与えられる. 具体的には, 以下の 4 つの手続きを基にコーパスを RDB に格納する.

(1) 各テーブル中の個々のエントリには, 一意の ID が付与される

(2) 直後に存在するといった線形順序関係や, どこどこの文節に係るといった非線型な順序関係は, 各エントリ へのポインタとして表記する

(3) 各階層構造の上位下位関係も同様に, 上位へのポインタ下位へのポインタという形で表現する

(4) 各テーブルは, (1) のための ID, (2),(3) を表現するためのポインタ群, さらに, そのテーブルに付与されたタグ (品詞, 読み, 原形 等) の列から構成される.

図 1 に, 係り受け関係が付与されたコーパス (具体的には京都大学コーパス [7]) を例に, 変換の概要を図示した. 図中の, 主辞と語形の定義は文献 [4] を参考にしている.

例えば,「形態素テーブル」中の「次形態素 ID」は, その形態素の直後に存在する形態素の ID が格納される. 同様に,「文節テーブル」中の「係り先 ID」は係り先の文節の ID が格納される. このように, 各エントリをポインタで表現することで, コーパスの階層関係と一対一に対応付ける事が可能になる.

3.3 SQL による柔軟な検索

図 1 の RDB から, SQL を用い, 実際に検索を行ってみる. ここでは検索に処理を絞るため, SELECT 文を中心に説明を行う. SELECT 文は,

```
SELECT 表示カラム
FROM 検索対象のテーブル（別名を付与できる）
WHERE 検索条件
```

という構文を持つ. 例えば,

```
SELECT 形態素
FROM morph
WHERR 品詞 = ’名詞’
```

というクエリは,「形態素テーブルから品詞が名詞である形態素を検索する」という意味になる.

さらに, 以下の例のように テーブル間の 結合操作を行う事で, より高度なクエリを作成する事ができる[4].

```
「奈良」を含む文を表示
SELECT sen1. 文, morph1. 形態素
FROM morph morph1, sen sen1
WHERE morph1. 形態素 = ’奈良’ and
      morph1. 文 ID = sen1.ID

「話す」を原形に持つ文節に係る文節と助詞を表示
SELECT morph2. 原形, morph3. 品詞,
       morph3. 形態素, seg1. 文節
FROM seg seg1, seg seg2,
     morph morph2, morph morph3
WHERE morph2. 原形 = ’話す’ and
      morph3. 品詞 = ’助詞’ and
      morph2. 文節 ID = seg2.ID and
      seg1. 係り先 ID = seg2.ID and
      seg1. 語形 ID = morph3.ID

品詞の bigram 統計の取得
SELECT morph1. 品詞, morph2. 品詞,
       COUNT(morph2.ID)
FROM   morph morph1 , morph morph2
WHERE  morph1. 次形態素 ID = morph2.ID
GROUP BY morph1. 品詞, morph2. 品詞
```

[4]同じ検索要求は, SELECT の入れ子 (副問い合わせ) により実現できる. しかし, 我々が使用している MySQL は副問い合わせをサポートしていないため, 結合操作で代用している.

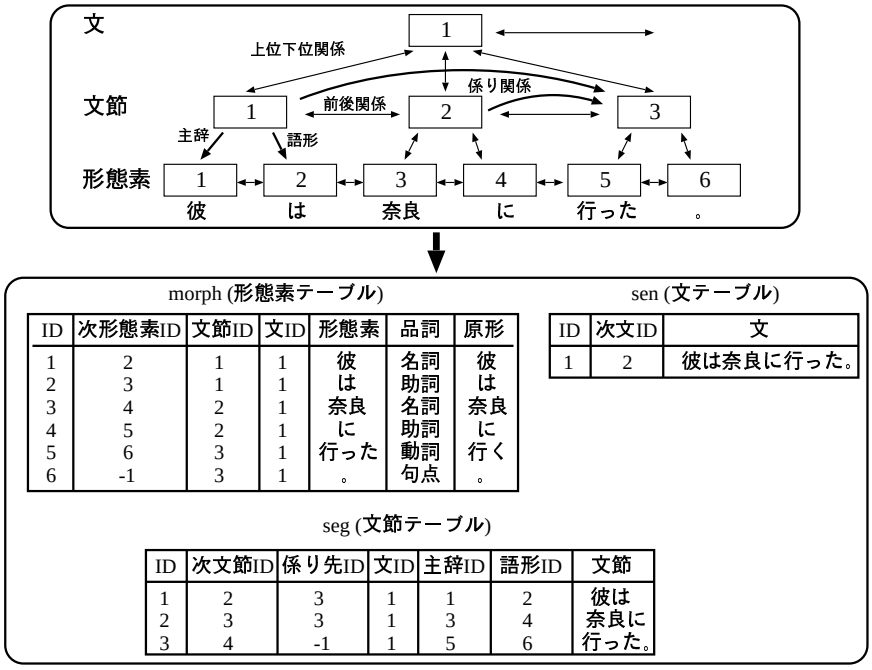


図 1: コーパスから RDB への変換

3.4 GUI ツール による SQL の生成

SQL を使用する事により, コーパス中の任意の部分木や連接関係, またそれらの頻度等を柔軟に検索する事ができる.

しかし, SQL のクエリを生成する事は, 熟練したテクニックが必要であり, 初心者には修得が困難である. また, 複雑なクエリになると, どのような検索を行っているのか直感的に分かりにくくなってくる.

そこで, 我々は クエリをより簡単に生成するための GUI ツール Query Builder [5]を作成した. このツールは, Microsoft Access のインターフェイスを踏襲しつつ, 言語処理に必要な頻度統計の集計機能等の機能を追加したツールである. RDBMS には MySQL[6]を使用している. また, ツールそのものは, Python[7] GTK[8] で記述され, 一般的な UNIX 上で動作する.

このツールを使う事で, SQL の知識が無くても, 比較的簡単な手続きのみでクエリを生成する事ができる. また, コーパス中の連接, 部分木, 階層関係等がグラフ表現として表示されるため, 複雑な検索式になってもその全体像が捉えやすい.

図 2 に, 起動直後のツールの図を示した. ツールは大きく 2 つの領域から構成される. 一つは, **関係エリア**, もう一つは**選択エリア**である.

- **関係エリア**
  コーパス中の部分木や連接といった, 各テーブルの関係を構築, 表示するためのエリアである. ここに, 任意のテーブルを配置し, Drag & Drop によって各テーブル間の ID やポインタの関係を記述していく.

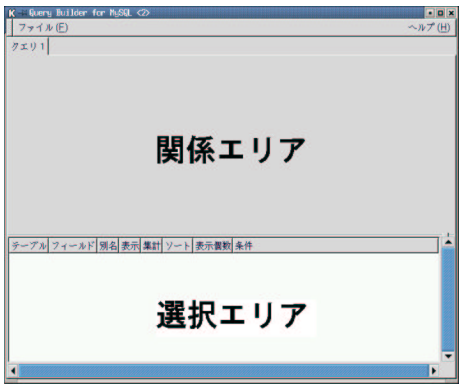


図 2: 起動直後の Query Builder

- **選択エリア**
  関係エリアで記述した関係の中で, 具体的にどのカラムを閲覧したいのかをここに記述する. 具体的には, テーブル中の任意のフィールドを Drag & Drop によってこのエリアにコピーする事で, そのフィールドが閲覧対象に加えられる. さらに, 選択エリアに加えられたフィールドに対し, 任意の条件付けが行える. 例えば, この形態素の品詞は「名詞」であるとか, 主辞の品詞は「動詞」といった条件付けがこれに該当す

[5]http://cl.aist-nara.ac.jp/~ taku-ku/software/qbuilder/
[6]http://www.mysql.com/
[7]http://www.python.org/
[8]http://www.gtk.org/

る．条件付けの操作として，一般的な文字列の完全一致をもちろん，前方，後方，部分一致，正規表現などがある[9]．

ここでは，以下の 2 つの例を取りあげながら，ツールの使用方法について解説を行う．

- 「奈良」が含まれる文を検索

  (1) 追加メニューをから morph, sen を選択し，関係エリアに moroh1 (形態素テーブル)，sen1 (文テーブル) を表示させる (図 3)．

  (2) 文中の形態素という関係を示すために，Drag & Drop によりリレーションを作成する (図 4)．

  (3) morph1 中の形態素が「奈良」であるという条件付けを行うために，moprh1 中の形態素フィールドを 選択エリアに Drag & Drop する．また文情報を表示する必要があるため，sen1 中の文フィールドを選択エリアに Drag & Drop する (図 5)．

  (4) 選択エリアの条件カラムをクリックし，実際に条件付けを行う．

  (5) 最終的に図 7 のクエリが完成される．検索結果として図 8 が得られる．

- 「を」と「に」格を持つ動詞の検索
  同様な手順により，図 9 のクエリを作成する．検索結果として 図 10 が得られる．
  この例の場合，実際には，以下のような SQL が生成される．これを一から手動で記述するのは熟練者であっても困難であろう．

```
SELECT seg1.文節, seg2.文節,
       seg3.文節, morph3.原形
FROM   seg seg1, seg seg2, seg seg3,
       morph morph1, morph morph2,
       morph morph3
WHERE (morph1.形態素 = 'を') and
      (morph2.形態素 = 'に') and
      (morph1.品詞 = '助詞') and
      (morph2.品詞 = '助詞') and
      (morph3.品詞 = '動詞') and
      (seg1.係り先 ID = seg3.ID) and
      (seg1.語形 ID = morph1.ID) and
      (seg2.語形 ID = morph2.ID) and
      (seg3.主辞 ID = morph3.ID) and
      (seg2.係り先 ID = seg3.ID)
```

さらに，詳細は省略するが，任意のフィールドをキーにして頻度統計を取ることも簡単に行える．例えば，品詞の bigram 統計等もさきほどの例と同様に，マウス操作のみで簡単に取得することができる．

[9]正規表現検索は，MySQL の拡張機能であるため，他の RDBMS ではサポートされないかもしれない．

図 3: 対象テーブルの選択

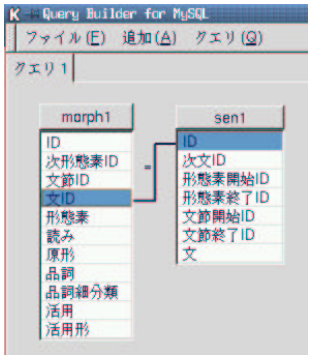


図 4: リレーションの記述

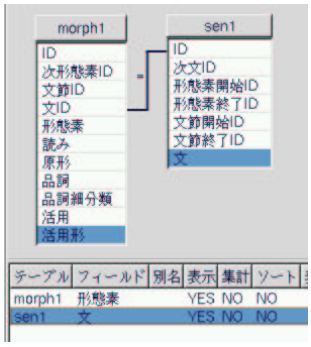
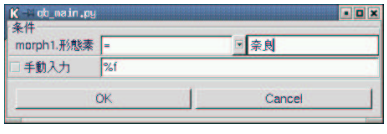


図 5: 選択フィールドの定義



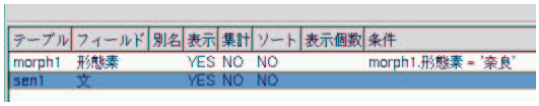


図 6: フィールドの条件付け

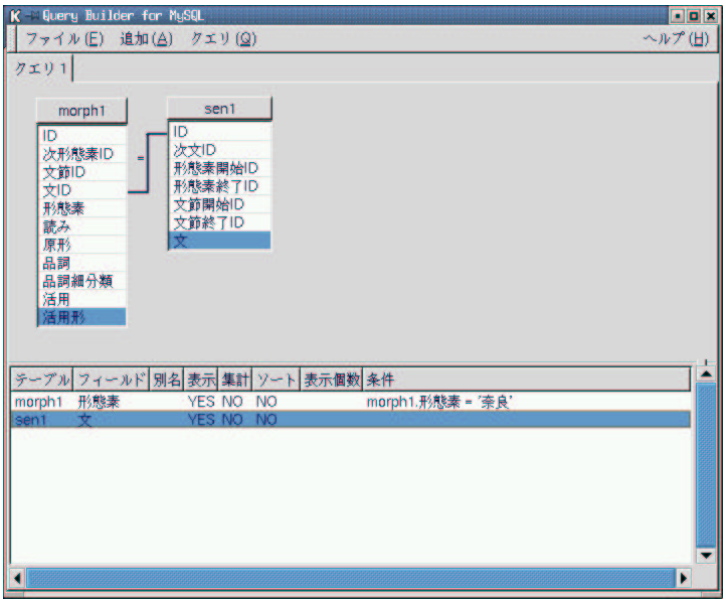


図 7: クエリの全体像 (例 1)

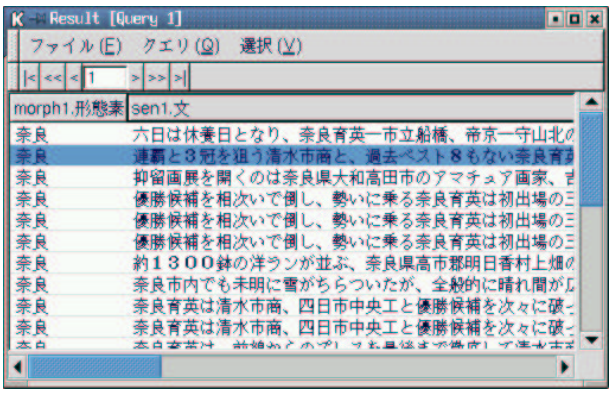


図 8: 検索結果 (例 1)

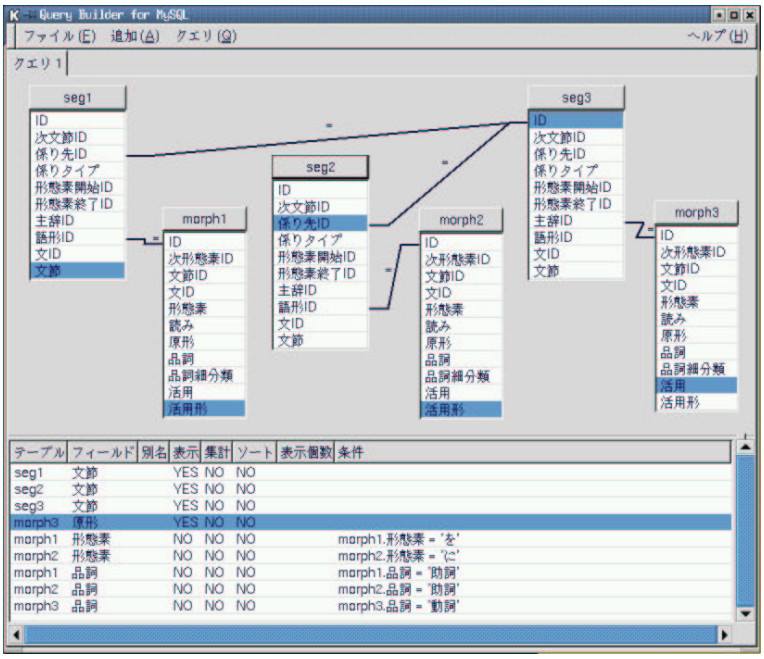


図 9: クエリの全体像 (例 2)

図 10: 検索結果 (例 2)

4 活用事例

Query Builder の使用対象としては, ある単語の使われ方の調査, 機械学習に与える学習データ抽出, 機械学習に与える素性の吟味等, 広範囲に渡ると予想される. 簡単なタスクならば, Query Builder 自身で完結する場合があり, 個々の処理のプロトタイピングとして使用する事も可能であろう.

実際に, 我々のグループは Query Builder を以下のような用途で活用している.

4.1 統計的係り受け解析の学習データ抽出

任意の二文節と, それに付随する周辺のコンテキストを抽出するクエリを作成すれば, その検索結果自身が統計的係り受け解析の学習データとして使用できる. あとは任意の学習アルゴリズムに検索結果を与えるだけでよく[10], 改めて素性選択のプログラムを作成する必要はない. 実際に文献 [2] では, Query Builder を用いて素性抽出を行なっている. また, 係り関係の個々の事例を閲覧する目的としても使用している.

4.2 日本語多義解消タスクにおける事例閲覧, 素性の抽出

日本語多義解消タスク [5] において, 多義性を解決する単語の周辺のコンテキストを閲覧する目的で使用している. 学習コーパスとしては, 我々が開発している ChaSen, CaboCha [11] を用いて, 形態素解析, 係り受け解析を行なったデータを使用した. 前後の文節や係り受け情報を含めた柔軟な検索が行えるため, タスクの概要を知る上で非常に役立つ. 一般には, 実際のデータを見る事無く, 直感的な判断のみで素性集合を構築しがちであるが, 実際のデータを簡単な手続きで閲覧する事が可能になるため, データに則した素性選択が行える.

4.3 コーパスの誤り検索

ChaSen の学習には, 現在 RWCP コーパスを使用している. コーパスの一貫性がどれだけ保たれているかで, 解析精度が大きく変わる. 解析結果に誤りが見つかると, 誤り付近のコンテキストを検索する事で, 誤りの原因追及が行える.

5 今後の課題

5.1 フォーマットの統一化

本稿では, 既存のコーパスを RDB を使って検索する事を中心に説明してきた. その一方で, このような検索環境を実現するためには, 既存のコーパスを RDB の内部データ構造に変換し, RDB に格納する必要がある. 具体的には, 各テーブルの要素に ID を付与し, 関連を持つエントリに正しいポインタを付与する作業の事を指す.

この処理は, 使用するテーブルの数が増えたり, 係り受け構造といった複雑な参照構造を付与されているコーパスの場合, 繁雑にならざるを得ない. さらに, 既存のコーパスのフォーマットが各々ばらばらであるため, 各コーパス毎にこれらの複雑な処理を実施する必要があり, RBB の構築を困難にしている.

今後は, RDB に格納する事を前提にしながら, 任意のコーパスを表現するための統一的かつ汎用的なフォーマットを規定したいと考える. 現状は, データベース中のテーブルやその中のフィールドの定義を各コーパス毎に個々に構築する必要があったが, この

[10] Query Builder には結果をファイルに出力する機能がある.
[11] http://cl.aist-nara.ac.jp/~taku-ku/software/CaboCha/

ような統一的なフォーマットを採用する事で, 個々に独立して行っていた変換作業を統一的な操作で行う事が期待できる. 具体的には, XML の利用などが考えられるだろう.

5.2 コーパス作成を含めた統合環境の構築

データ主導型の言語研究には, 充分な量のコーパスの存在が重要である. 一般に, 整合性を保ちながら一貫性のあるコーパスを人手により作成する事は極めて困難な作業である.

本稿で紹介したコーパス検索ツールは, このようなデータの整合性を保つ目的として十分に適用可能と考える. 例えば, コーパスを作成する過程において, 一貫性を保つために過去に付与したタグの情報を検索し, ユーザに提示したり, 現在付与したタグが一貫性があるものなのか, 自動的にクエリを作成しチェックする機能がそれに該当する.

現状はコーパスの検索のみに特化したツールであるが, 今後はコーパスの構築を含めた統合環境として機能を拡充させたいと考える.

6 まとめ

RDB を用いる事で, コーパス中の任意の部分木や連接関係, またそれらの頻度を柔軟に検索できる事を示した. また, 検索式を構築するための, GUI ツールを提供し, SQL の知識が無くても, 直感的な操作で検索式を構築する環境を構築した.

参考文献

[1] Laura Kallmeyer. A query tool for syntactically annotated corpora. In *Empirical Methods in Natural Language Processing and Very Large Corpora*, pp. 190–198, 2000.

[2] Taku Kudoh and Yuji Matsumoto. Japanese Dependency Structure Analysis Based on Support Vector Machines. In *Empirical Methods in Natural Language Processing and Very Large Corpora*, pp. 18–25, 2000.

[3] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. In *1st ACM-SIAM Symposium on Discrete Algorithms*, pp. 319–327, 1990.

[4] 内元清貴, 関根聡, 井佐原均. 最大エントロピー法に基づくモデルを用いた日本語係り受け解析. 情報処理学会論文誌, Vol. 40, No. 9, pp. 3397–3407, 1999.

[5] 白井清昭, 橋田浩一, 有田英一, 井佐原均, 荻野紫穂, 柏野和佳子, 小船隆一, 高橋裕信, 徳永健伸, 長尾確, 橋本三奈子, 村田真樹. 岩波国語辞典を利用した語義タグ付きテキストデータベースの作成. 情報処理学会研究会報告 200-NL-141, pp. 117–124, 2000.

[6] 山下達雄, 松本裕治. Suffix array を用いたフルテキスト類似用例検索. 情報処理学会研究会報告 97-NL-121, pp. 83–90, 1997.

[7] 黒橋禎夫, 長尾眞. 京都大学テキストコーパス・プロジェクト. 言語処理学会 第 3 回年次大会, pp. 115–118, 1997.